MULTIAGENT PLANNING, CONTROL, AND EXECUTION

Chapter 11 of
Multiagent Systems: A Modern Approach
http://www.the-mas-book.info

E. Durfee and S. Zilberstein

Planning

- Necessary when near-term choices of actions can enable, or prevent, later action choices required to achieve goals.
- Possible when agent possesses a sufficiently detailed and correct model of the environment, and of how actions affect the environment.
- Challenging because the space of possible plans grows exponentially with the plan duration.

Multiagent Planning

- Now the near-term choices of actions can enable, or prevent, later action choices *of others* required to achieve goals, and *others'* near-term actions can affect the agent's later choices too.
- Possible when agents can explicitly or implicitly model others' plans, and predict outcomes in the environment of executing the plans jointly.
- Challenging because the space of possible individual plans grows exponentially with the plan duration, and of multiagent plans grows exponentially in the number of agents.

Multiagent Planning, Control, and Execution

- Assuming that agents are cooperative:
  - Strive to maximize some joint performance measure
- Agents' planned activities should dovetail well to maximize achievement of joint objectives.
- Agents' immediate control decisions should jointly contribute to improving collective state.
- Agents should monitor outcomes of joint actions and progress of joint plans to execute as a responsive multiagent team.

Problem Structure: Composition

- Each agent's state is factored:
  - State is represented as a set of features
  - A feature might only be affected by particular actions
- Multiagent state is also factored:
  - Different agents will perceive, and be able to change, different (but possibly overlapping) features of the joint environment
  - Some features might be purely local to a particular agent.

Problem Structure: Locality

- Efficient single-agent planning/control relies on assumptions of locality:
  - An action only affects a (small) subset of state features
  - Most of the state features are unaffected by any particular action
- Efficient multiagent planning/control extends the locality assumption:
  - An action taken by one agent only affects a limited number of other agents' states
  - As well as only a localized subset of each of their state features
  - Hence, agents are loosely- (aka weakly-) coupled

What Aspects Are Multiagent?

- Multiagent planning/control could refer to just the product of the planning/control process:
  - A centralized process builds a plan/control representation that specifies how each of multiple agents should behave
- Multiagent planning/control could refer to the process of formulating plan/control decisions:
  - Multiple agents participate in the construction of a single plan or control policy

What Aspects Are Multiagent?

- Both the product and the process are multiagent:
  - Each agent applies its local expertise and awareness to construct its local plan.
  - Agents use communication, and/or shared knowledge and biases, to shape their local plans to conform better to others' plans, in order to more effectively achieve collective objectives.

Flavors of Multiagent Planning/Control

- Coordination prior to local planning/control
  - Committing to how to work together, and then making suitable local planning/control decisions
- Local planning/control prior to coordination
  - Formulating local plan/control decisions separately, then adjusting them for coordination
- Decision-Theoretic Multiagent Planning
  - Multiagent planning in the face of non-determinism and partial observability
- Dynamic multiagent planning/control
  - Monitoring and replanning during execution

Coordination Prior to Local Planning

- Formulate interaction plans/rules beforehand, and commit to following them
  - Example: Message-exchange protocol defining interpretations of and allowable responses to (sequences) of communicative acts
- Main ideas:
  - Core aspects about what coordination decisions will need to be made and how they will be resolved are known ahead of time
  - Details of agents' plans specific to a particular problem instance can fit into the predefined coordination framework

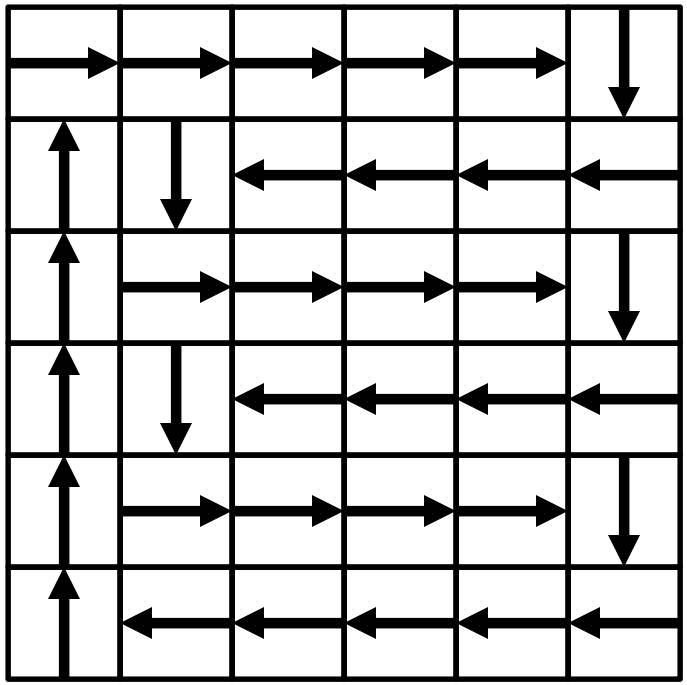
Social Laws

- Basic idea: Identify joint states that should be avoided, and impose restrictions (laws) on agents' action choices to prevent them.
- Canonical example: Avoid collisions.
  - Mobile robots moving in an open space risk colliding with and disabling each other.
  - Yet, centrally controlling all of their motions is overkill: micromanaging largely-independent behaviors, potential single point of failure, and scales poorly as number of robots grows

Social Laws for Collision Prevention

- First pass: Never enter a location that is occupied.
  - Does not account for simultaneous movement, when more than one robot enters the same (previously) unoccupied location.
- Second pass: Restrict direction from which a location can be entered to only one choice.
  - Now, collision cannot occur if world begins in a “safe” state.
  - Creates equivalent of “one-way” locations.
  - Need to be careful in the creation of these to ensure that every location can eventually be reached.

Social Laws in Grid Environment



- So long as agents start in different locations, and keep moving at every step in the dictated direction, then eventually each can transit between any pair of locations.
- Hence, they can independently plan the order of picking up and dropping off items, making all deliveries without fear of collisioin.
- But distance traveled will generally be larger than the minimum necessary.

Conventions

- Flip side of social laws to encourage, rather than prohibit, particular outcomes
- Basic idea: Identify joint states that are preferred, and impose restrictions (laws) on agents' action choices so as to achieve them.
- Canonical example: Shared awareness of goals.
  - If an agent that is cooperating on a goal with others comes to believe that achieving the goal is impossible,
  - Then rationality would dictate that it stop taking actions associated with achieving the goal,
  - And by convention must inform the cooperating agents so that they also avoid wasting effort.

Social Laws/Conventions Formation

1. Identify joint states that should be avoided (or sought).
2. Work backward through agents' joint actions to identify possible precursor states to the states to avoid (seek).
3. Impose constraints on agents' choices of actions in the precursor states to prevent (or require) reaching the states to avoid (seek).
4. Recurse: If no action choices exist in a precursor state that avoid (attain) the target state, then the precursor state itself becomes a state to be avoided (sought).

Some Social Laws/Conventions Challenges

- Identifying prior to agent execution all of the states that should be avoided (sought).
  - When state space is large/infinite, some bad (good) states might be missed.
  - New laws/conventions could need to be legislated dynamically in response to unexpected outcomes.
- Different laws could achieve the same (safety) results but have significant impacts on performance.
  - E.g., Cars drive east-west on even days, north-south on odd
- Pushed further, then performance might get even better if different laws apply to different agents.
  - E.g., Fire trucks have authority to violate some laws

Organizational Structuring

- Basic idea: Assign complementary roles to agents, where agents' roles bias their choices of actions and lead to better cooperative behavior.
- Canonical example: Sensor networks.
  - Agents in different locations are responsible for monitoring complementary regions,
  - Some agents might have particular responsibility for fusing sensed data/interpretations of others.

Organizational Structuring

- Basic idea: Assign complementary roles to agents, where agents' roles bias their choices of actions and lead to better cooperative behavior.
- Canonical example: Sensor networks.
  - Agents in different locations are responsible for monitoring complementary regions,
  - Agemts with different sensing modalities are responsible for detecting particular phenomena,
  - Some agents might have particular responsibility for fusing sensed data/interpretations of others.

Organizational Design

- Encode in computational form the roles/protocols exhibited by humans.
- Detect patterns of recurrent interactions among agents working from first principles, and compile these into roles and protocols.
- Decompose the task from the top down, and base roles on subtasks and interactions on subtask relationships.
- Define a space of organizational designs, and search over that space, using expectations about the task-environment to evaluate alternative (partial) organizational designs.

ORGANIZATIONSEARCH

Initialize space of candidate partial organizations with a single candidate with the overall goal as its single (leaf) node.

1. Generate expansions of a candidate partial organization by finding an unbound goal leaf in its decomposition hierarchy, and replacing it either with a role-goal binding (e.g., assigning it to an agent capability), or with a subgoal tree that decomposes it further.
2. Repeat step 1 until all leaves have associated roles.
3. Use information about available agents' capabilities to assign agents to the roles.
4. If all roles assigned, return organization; else, backtrack to try different decompositions.

Organization Execution

- A purpose of adopting an organization is to simplify the operational control decisions of the agents:
  - The organizational biases should lead agents towards compatible activities, so an agent following its role specifications should not need to model what others are doing.
- This can lead to good, but suboptimal, performance:
  - Organizational inefficiencies can arise from lack of detailed awareness of concurrent activities across the organization.

Functionally-Accurate Cooperation

- A characterization of organizational performance:
  - Agents' decisions lead to accurate functioning of the collective, in the limit, even though at any given time the system might not be completely accurate.
  - Agents' actions cooperatively lead to proper functional performance, even though no single agent can achieve that performance alone.
- FAC posits that, given enough time and information exchange, agents will eventually converge on good global solutions.

FAC Applications

- FAC is appropriate for cognitive tasks where multiple, tentative, partial solutions can be considered at once:
  - Interpretation of sensor network data.
  - Design of an artifact, plan, process, etc.
- FAC agents typically utilize architectures that expedite the efficient storage and retrieval of tentative partial solutions:
  - E.g., Blackboard architectures.

FAC Agent Interactions

- Exchange of tentative partial solutions increases:
  - Completeness of solutions (combination).
  - Confidence in solutions (corroboration).
  - Precision of solutions (refinement).
- Uncontrolled, exchange can engender distraction and duplication of effort.
- For this reason, FAC often combined with organizational structuring:
  - Agents' roles and responsibilities bias their decisions about what to exchange, what to work on locally, etc.
  - Communication protocols guide timing decisions about how locally-complete a hypothesis should be before it is shared with other agents

Agent Interaction Variations

- Rather than voluntarily exchanging information unprompted, protocol could be request-driven:
  - Agent identifies characteristics of information that would be helpful to have, and queries others for it.
  - Whom to ask can be guided by organizational knowledge.
  - Asking can in fact influence behavior of asked agents to prioritize finding an answer.
  - Can be more efficient, but introduces more delay in information exchange (2 rounds of message passing instead of 1).
- Reducing rounds of iterative communication can also be accomplished by conveying multiple tentative hypotheses in the same round:
  - E.g., several available times for scheduling a meeting.
- Repetition (“murmuring”) for undependable channels.

Result-Sharing vs. Task-Sharing

- FAC with Organizational Structuring assumes that agents' different roles inherently distribute tasks among them, so joint problem solving involves sharing partial, tentative results.
- A common alternative of moving results to agents whose tasks (roles) need them, is to instead move tasks to agents that can do them.
- That is, cooperative problem solving involves identifying how and where agents should share tasks such that tasks are assigned to agents that are best able to do them.

Task-Sharing Protocols

- A protocol in this context represents a template for a pre-defined plan:
  - To achieve the goal of assigning tasks/roles to agents most able to do them;
  - The protocol provides a communication plan template for the agents to follow;
  - Where the specific tasks/roles to be done, and how agents can express their suitability for doing them, can vary.
- Formulating protocols is similar to social laws and organizational structures:
  - Identify desirable states of the world (e.g., tasks/roles distributed well);
  - Identify patterns of actions that if jointly followed will bring those states about.

Contract-Net Protocol Example

- The first well-studied multiagent protocol.
- Investigated in the context of distributed sensor net establishment (DSNE):
  - Given high-level goal of monitoring a region;
  - Decompose overall monitoring objective into a set of smaller roles (e.g., regions to monitor, fusion of results from different regions).
  - Discover agents whose positions and/or resources permit them to perform the roles, and assign the roles accordingly.

Contract-Net Protocol Process

1. An agent whose role/task exceeds its abilities decomposes the role/task into pieces that, if all performed, achieve the desired performance.
2. For each subrole/task, this Manager agent initiates the contracting protocol:
   1. It formulates a task-announcement message describing the task, the capabilities required of agents eligible to accept the task, and the contents of a bid for the task.
   2. It broadcasts the message, or if it has knowledge about which agents are likely candidates it can address the announcement just to them.

   In the DSNE domain, an announcement message could indicate the regional coverage needed by whomever takes on the role, and the bid specification might request a summary of the sensory capabilities/limits of the potential recipient of the role/task.

Contract-Net Protocol Process (2)

3. A potential Contractor agent receiving an announcement message:
   1. Confirms that it satisfies the eligibility requirements;
   2. Uses the task/role description to determine the degree to which is it willing and able to perform the role/task;
   3. Generates and submits a bid in the specified format.
4. The Manager agent :
   1. Collects the bids sent by the contractors.
   2. If no (acceptable) bids are received, it sends out a revised announcement (relaxing eligibility requirements, or modifying the expectations of the role/task.
   3. If acceptable bids are received, it accepts one (or more, if redundancy is needed for robustness) and awards the task/role.

In the DSNE domain, the Manager might need to carve up the region differently to better match the spatial arrangement of existing sensors. If the right placements of sensors are available, it will decide which sensor in each subregion is “best” (is most reliable, powerful, available…) and assign the roles accordingly.

Contract-Net Protocol Process (3)

5. A winning Contractor adopts the assigned role/task, which could require that it withdraw bids sent to other managers.
6. It performs its role/task:
    1. It could further decompose its task, recursively invoking the Contract-Net protocol
    2. It could send interim reports back to the Manager reporting on its progress/results so far.
7. When the task/role is completed, it sends final information to the Manager.
8. Upon receiving reports, the Manager:
    1. Combines reports from the different Contractors to synthesize a full view of the (tentative) global solution.
    2. Cancels or redirects Contractors if collective performance is not on a satisfactory trajectory.

In the DSNE domain, Contractors will monitor their assigned regions, sharing partial maps of the phenomena detected in their regions with the Manager, which will compose these results into a global view of the phenomena.

Tradeoffs About When to Coordinate

- Coordination first, like in social laws and organizational structuring, decouples agents' local problems so that they can plan (and replan) independently.
- But, as has been seen, imposing the coordination constraints might be overkill for any particular problem instance:
  - Laws unnecessarily restrictive
  - Organizational inefficiencies.
- Alternative is to wait until agents know what they want to do, and then coordinate their particular plans rather than coordinating for all possible plans.

Local Planning Prior to Coordination

- Appeals to locality and decomposability arguments
  - That the collective endeavor is composed of largely independent activities done by individuals.
  - And that interdependencies are local to small numbers of individuals.
- This argues for a divide-and-conquer approach:
  - Each individual plans as if it were completely independent.
  - Then any interdependencies are identified and resolved.

Multiagent Plan Coordination Problem (MPCP)

- Finds a multiagent plan that is a combination of agents' local plans, adjusted to account for interdependencies.
- Resulting plan could differ from a multiagent plan that considers the full space of joint actions.
- Has a distributed constraint satisfaction flavor:
  - Assignments of variables (plans or pieces of plans) that satisfy local and interagent constraints.
  - But domain of variables (plan spaces) are too large to enumerate, and constraints can be expensive to check.

Basic MPCP Approach

1. Each agent builds its own plan as if it were alone.
2. Agents directly, or through a more centralized intermediary, identify potential conflicts/inefficiencies that could arise during joint execution.
3. To resolve such problems, agents inject additional constraints (for example, semaphores to prevent bad combinations of actions).
4. If all problems prevented, then done. Else, one or more agents formulates an alternative local plan and the process repeats.

State-Space MPCP Approach

- Detects problems in joint execution by projecting forward through plan execution, in a graph-planning manner.
  1. From current state, consider combination of actions consisting of the next action of each agents' plan.
  2. Using mutex concepts, identify impermissible combinations of actions.
  3. Impose timing constraints that prevent mutex actions, postponing some of them, to create a legal next "current" state, and repeat the process.
- The process above can search over different choices of which actions to postpone to ultimately find a joint execution sequence that achieves the agents' combined goals.

Plan Combination Search

Ephrati & Rosenschein, AAAI 1994

- Variation on state-space MPCP techniques.
- Each agent starts with a space of plans that can achieve its goals.
- Mutex can rule out particular combinations of agents' plans, but least-commitment of maintaining a space of plans supports finding better (nearly optimal) joint plans.
- A* search technique to explore alternative paths through action combinations, where heuristic includes estimated further costs from a particular joint state to a goal-satisfying state.

Plan-Space MPCP Approach

- Instead of using agents' individual plans to search through the spaces of joint states that they might induce;
- Search through a space of joint plans of the agents.
- Builds on single-agent planning techniques, and in particular partial-order causal-link (POCL) planning.

Single-Agent POCL Plan

- A partial-order causal-link plan is defined as:

Definition 4.1 *A **POCL plan** is a tuple $P = \langle S, \prec_T, \prec_C \rangle$ where S is a set of plan steps (operator instances), $\prec_T$ and $\prec_C$ are (respectively) the temporal and causal partial orders on S, where $e \in \prec_T$ is a tuple $\langle s_i, s_j \rangle$ with $s_i, s_j \in S$, and $e \in \prec_C$ is a tuple $\langle s_i, s_j, c \rangle$ with $s_i, s_j \in S$ and where c is a condition. A POCL plan models the agent's initial state using an* init *step, $init \in S$, and the agent's goal using a* goal *step, $goal \in S$, where $post(init) = I$ (the initial state conditions), and $pre(goal) = G$ (the goal conditions).*

Single-Agent POCL Planning

1. Initialize the plan with the init and goal steps.
2. While there is a flaw (a causal-link conflict or an open precondition):
    1. Select a flaw to eliminate.
    2. Eliminate that flaw:
        - Add ordering constraints to resolve conflict.
        - Add causal link (and a new step creating it if needed) to resolve open precondition.
3. When no flaws remain, return plan.

Example Single-Agent POCL Planning Problem

Blocks world with blocks A, B, C, and D

Goal: Block A should be on block B.

Initial state:

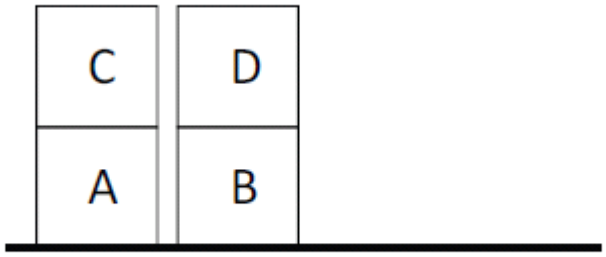


Figure 11.1: Initial state for simple blocks-world problem.

Example Single-Agent POCL Planning Solution

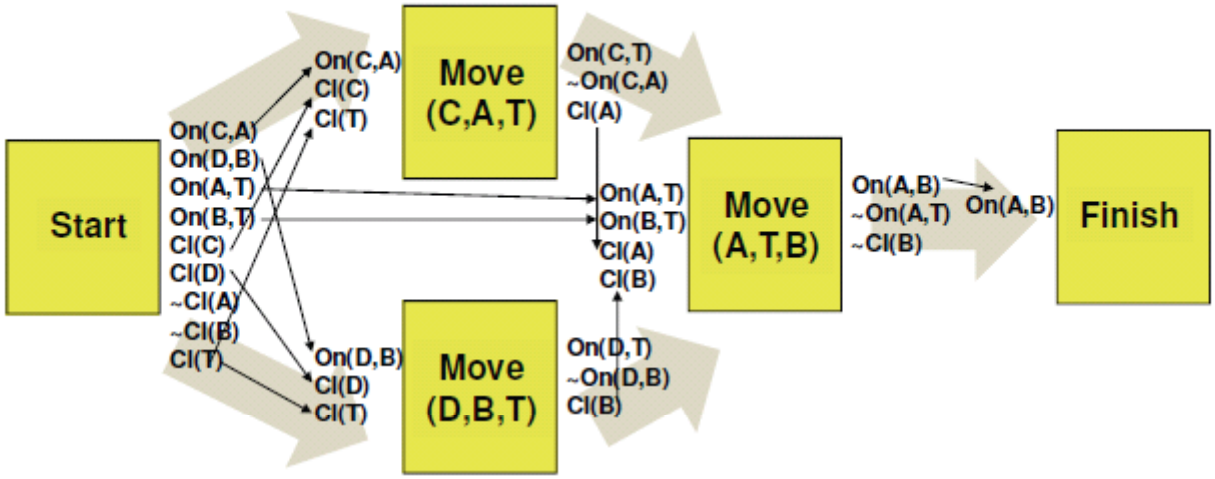


Figure 11.2: Single POCL plan for stacking block A on block B.

- 3 new steps introduced.
- Moving C from A to Table, and D from B to Table, are unordered wrt each other, but both must precede moving A from Table to B.

Different Single-Agent POCL Plan

- Same initial state, but goal is that block B should be on block C.
- Requires adding 2 new steps, and creates a totally-ordered plan.

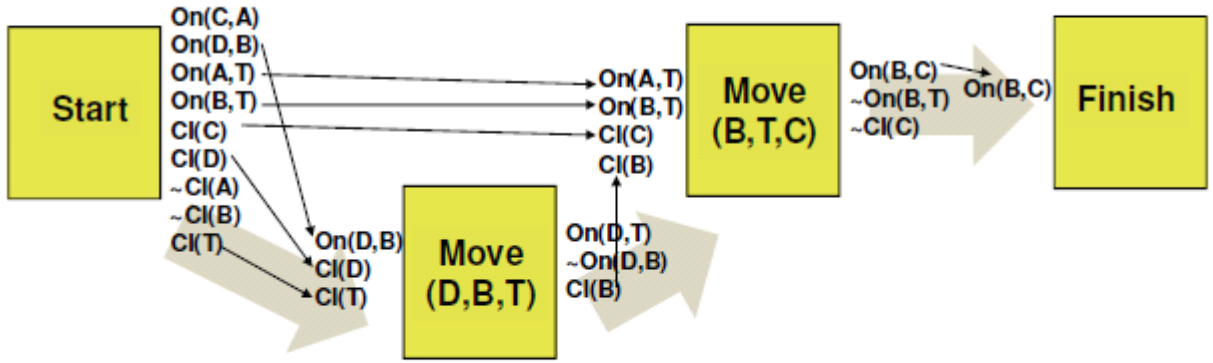


Figure 11.3: Single POCL plan for stacking block B on block C.

Parallel POCL Plan

- Whether single-agent or multiagent, need to account for possibility that more than one action can happen at a time.
- Plan specification needs to express whether steps must be taken simultaenously, or must not be taken simultaneously.

Definition 4.2 *A **parallel POCL plan** is a tuple* $P = \langle S, \prec_T, \prec_C, \#, = \rangle$ *where* $\langle S, \prec_T, \prec_C \rangle$ *is the embedded POCL plan,* $\#$ *is a symmetric* non-concurrency relation *over the steps in* S*, and* $=$ *is a symmetric* concurrency relation *over steps in* S.

- This introduces another type of possible flaw, corresponding to mutually-exclusive steps due to inconsistent effects:

Definition 4.3 *A **parallel-step conflict** exists in a parallel plan when there are steps* s_j *and* s_i *where* $post(s_i)$ *is inconsistent with* $post(s_j)$, $s_j \not\prec_T s_i$, $s_i \not\prec s_j$ *and* $\langle s_i, s_j \rangle \notin \#$.

Multiagent POCL Plan

A parallel POCL plan that expresses the assignments of which agents are responsible for which steps:

Definition 4.4 *A **multiagent parallel POCL plan** is a tuple* $M = \langle A, S, \prec_T, \prec_C, \#, =, X \rangle$ *where* $\langle S, \prec_T, \prec_C, \#, = \rangle$ *is the embedded parallel POCL plan,* A *is the set of agents, and* X *is a set of tuples of form* $\langle s, a \rangle$*, representing that the agent* $a \in A$ *is* assigned *to execute step* s*. A multiagent plan models the agents' initial states using* init *steps,* $init_i \in S$*, and the goals of the agents using a set of goal steps,* $goal_i \in S$ *where the preconditions of the* goal *steps represent the conjunctive goal that the plan achieves, and the postconditions of the* init *steps represent features of the agents' initial states before any of them take any actions.*

Uncoordinated Multiagent POCL Plan

Combines the 2 single-agent plans.

Requires that both initial states, and both goal states, be concurrent.

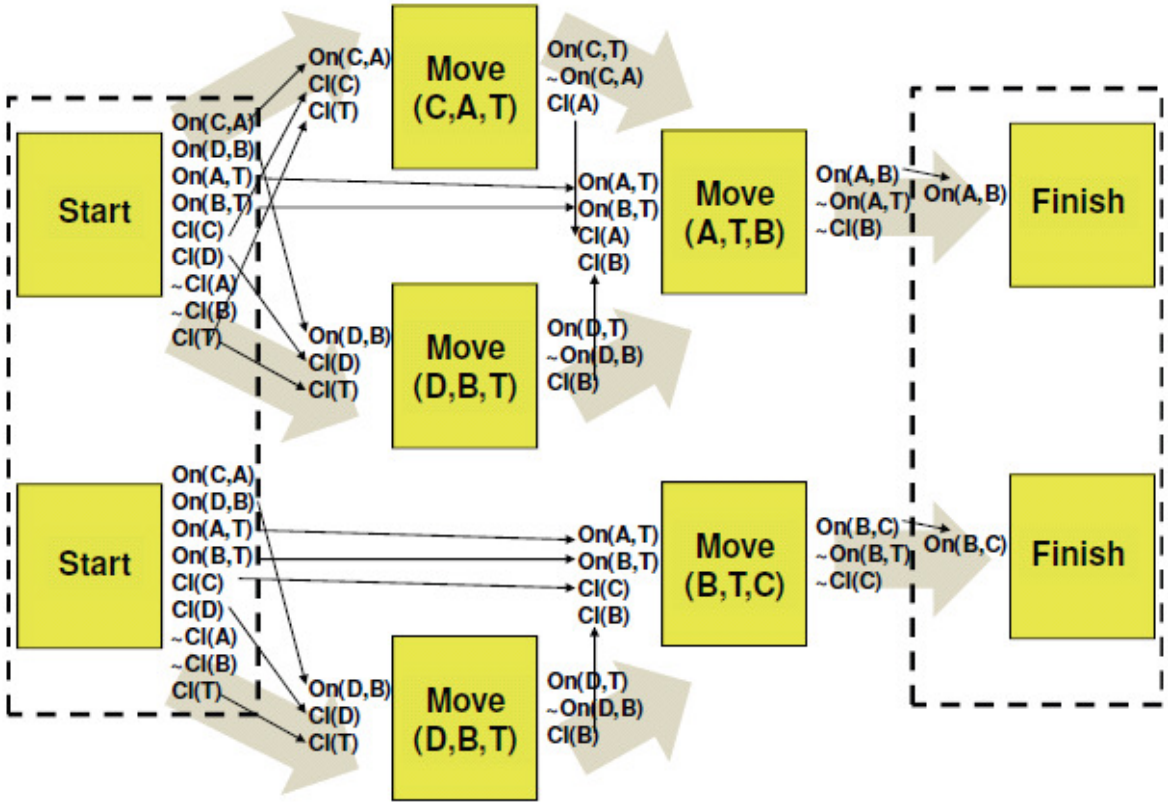


Figure 11.4: Initial (inconsistent) multiagent parallel POCL plan.

Multiagent Plan Coordination Problem

Rearrange ordering constraints and causal links so as to resolve all flaws;

<u>Without adding any new steps.</u>

Example: Move(A,T,B) threatens Cl(B) causal link between Move(D,B,T) and Move(B,T,C).

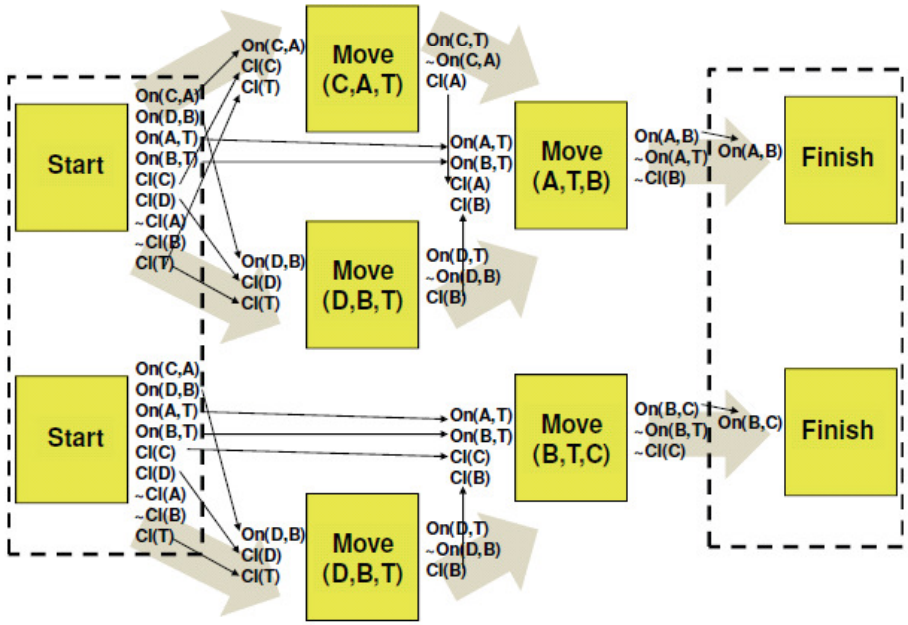


Figure 11.4: Initial (inconsistent) multiagent parallel POCL plan.

Multiagent Plan Coordination Process

The combined agents' plans represent a (flawed) plan:

Utilize the standar POCL planning algorithm of finding and repairing flaws.

Example: Adding a temporal ordering constraint resolves threat.

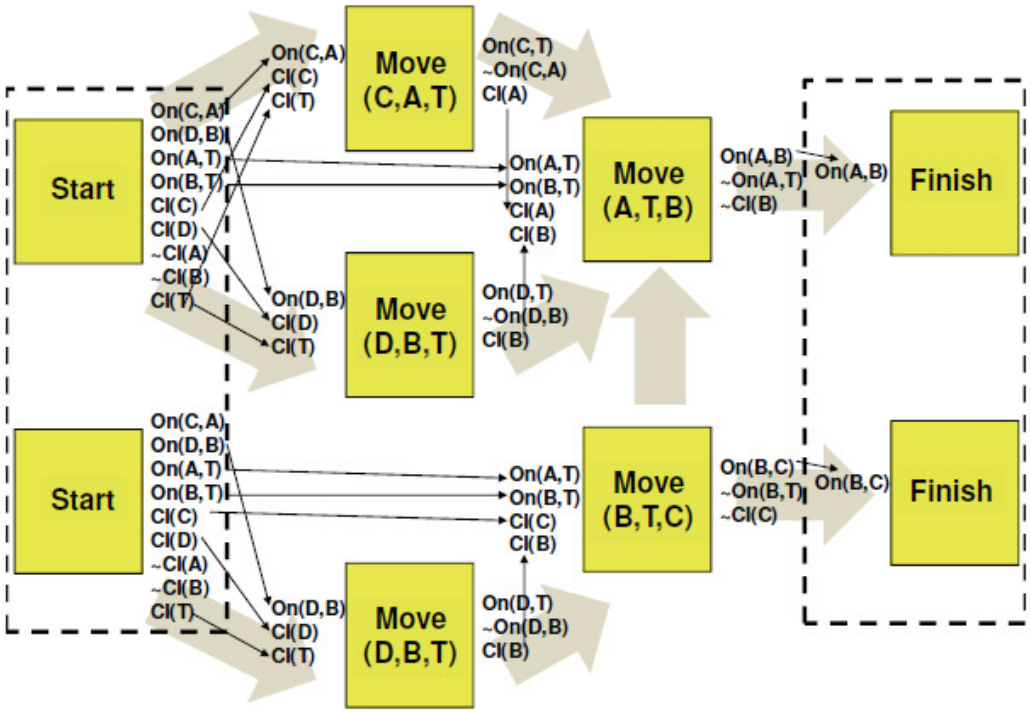


Figure 11.5: Ordering constraint added to resolve causal-link threat.

Redundancy Flaw

Even with causal-link threat resolved, resulting multiagent plan arguably still is flawed:

- Redundant "Move(D,B,T)" actions could lead to misbehavior (collision at block; an agent doing the move, and then the other putting D back on B so that it can do the move too…)

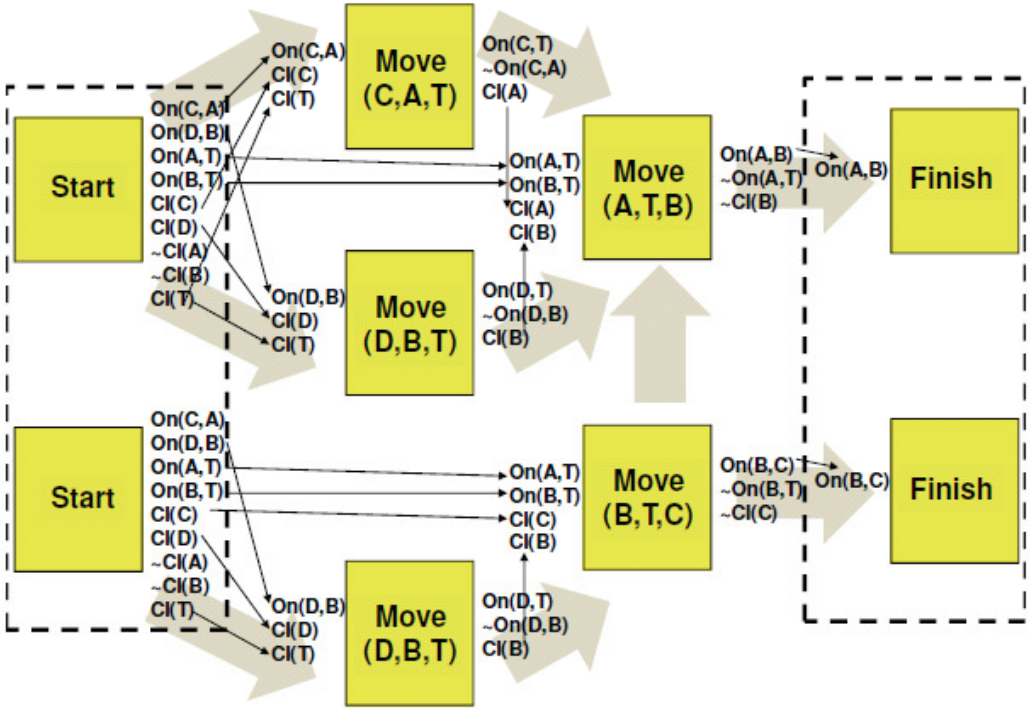


Figure 11.5: Ordering constraint added to resolve causal-link threat.

Plan Step Redundancy

Explicitly represent and resolve this type of flaw.

Definition 4.6 *A plan step s is **redundant** in a multiagent parallel POCL plan M with steps S when there exists a set of replacing steps R, where $R \subseteq S$, such that for each causal link of form $\langle s, s'', c \rangle$, it is also the case that $\exists s' \in R$ s.t. $c \in post(s')$.*

Redundancy flaws can be repaired by searching for a way to redirect the causal links coming from one step so that all instead come from other existing steps.

Fully Coordinated Multiagent POCL Plan

All flaws have been resolved:

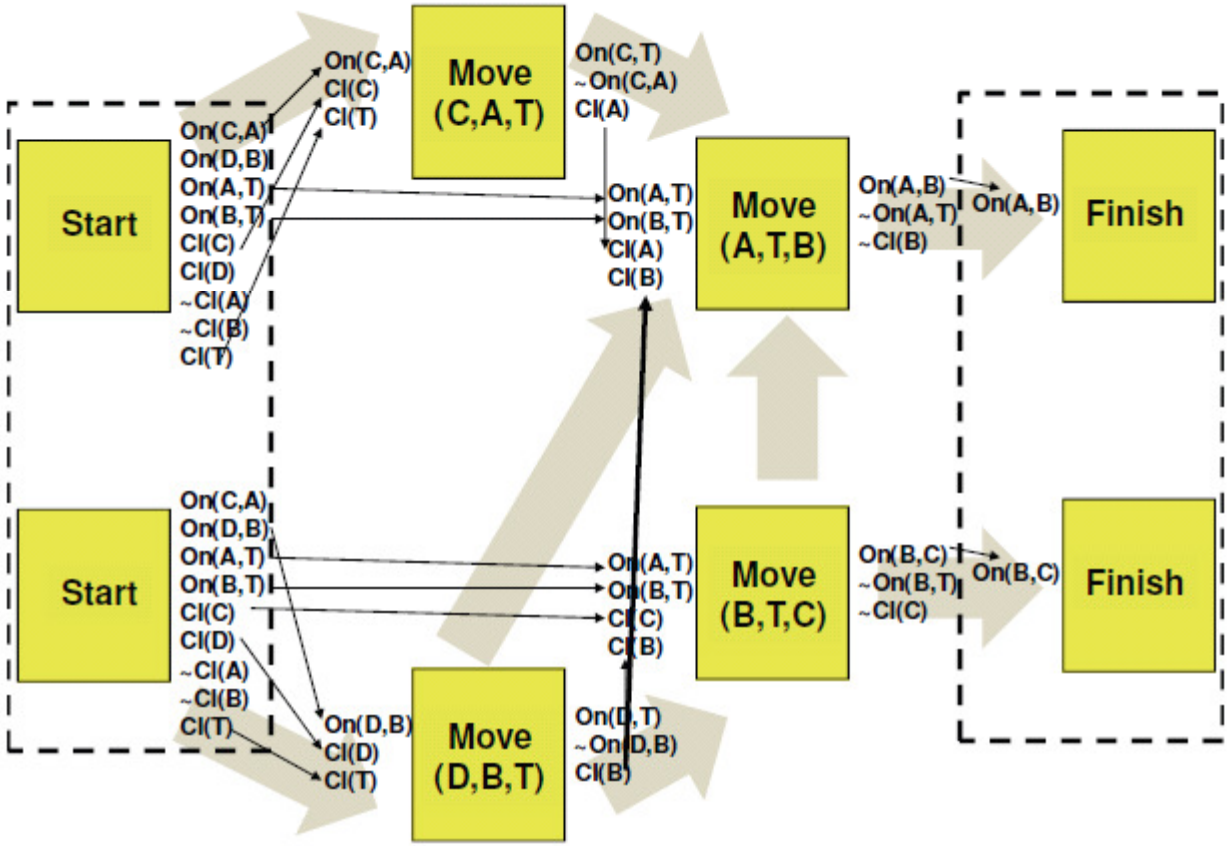


Figure 11.6: Solution multiagent parallel POCL plan.

Multiagent Plan Coordination by Plan Modification Algorithm

Algorithm 1: Multiagent Plan Coordination by Plan Modification Algorithm

Input : an (inconsistent) multiagent parallel plan
Output: an optimal and consistent multiagent parallel plan or null plan
Initialize *Solution* to null;
Add input plan to search queue;
while *queue not empty* **do**
    Select and remove multiagent plan M from queue;
    if *M not bounded by Solution* **then**
        if *(M passes Solution Test)* **and** *(steps in M < steps in Solution)* **then**
            $Solution = M$;
        end
        Select and adjust a non-flagged causal-link in M;
        For each refinement, remove unnecessary steps in plan;
        Enqueue all plan refinements in search queue;
    end
end
repair parallel-step conflicts in *Solution*;
return *Solution*;

Hierarchical Planning

- A plan-space planning approach:
  - Incrementally creates a plan by refining more abstract plan steps, expanding them into more detailed subplans.
  - Exploits knowledge captured in the form of a library of subplans: rather than constructing a plan directly as a search through primitive executable actions, retrieve and combine subplans that have been prebuilt to achieve typical (sub)goals.
  - The planning process is complete when the plan is refined down to the level of executable actions.

Multiagent Hierarchical Planning

- Extends the notion of an abstract plan step:
  - Not only abstracting over time, but also over performer of actions.
  - Team-oriented programming mindset: Abstract actions can correspond to a group activity, where refinements of those actions break down the different roles/tasks of agents.
  - In this sense, very much like organizational structuring:
    - Differs in the Organizational Structure is assumed to persist over multiple problem instances, where multiagent hierarchical planning would formulate a plan for a particular problem instance.

Hierarchical Coordination

- Basic ideas:
  - Modifications to agents' individual plans to achieve coordination don't have to be done at the most primitive level.
  - Plans at abstract levels are smaller and simpler, making coordination easier.
- Strategy:
  - Work downward from abstract plans to discover and resolve flaws.
  - Decide whether to resolve a flaw at an abstract level (potentially introducing more constraints than needed) or to expend the effort to look for how the flaw manifests at a more detailed level, to restrict coordination constraints more narrowly where needed.

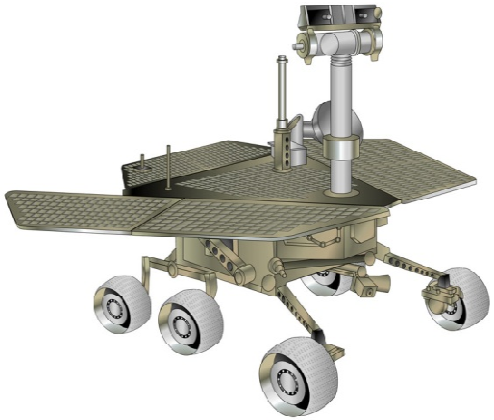
Hierarchical Behavior-Space Search

1. Initialize the current-abstraction-level to the most abstract level.
2. Agents exchange descriptions of the plans and goals of interest at the current level.
3. Remove plans with no potential conflicts. If the set is empty then done; else decide whether to resolve conflicts at the current level or at a deeper level.
4. If conflicts are to be resolved at a deeper level, set the current-abstraction-level to the next deeper level and set the plans/goals of interest to the refinements of the plans with potential conflicts. Go to step 2.
5. If conflicts are to be resolved at this level:
   a) Agents form a total order. Top agent is the current superior.
   b) Current superior sends down its plan to the others.
   c) Other agents change their plans to work properly with plan of current superior, without introducing new conflicts with past superiors.
   d) Once no further changes needed in plans of the inferior agents, the current superior becomes a previous superior and the next agent in the total order becomes the superior. Loop back to step b. If there is no next agent, then the protocol terminates and agents have coordinated their plans.

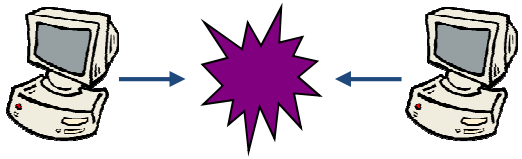
Decision-Theoretic MA Planning

- A **group of agents** interact in a stochastic environment
- Each “episode” involves a **sequence** of decisions over some **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 obtains **different partial observations** of the overall situation
- Decision makers have the **same objectives** characterized by a **single reward function**

Applications



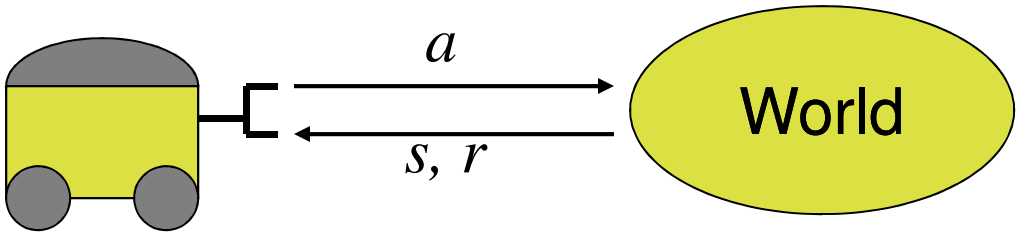
- Autonomous rovers for space exploration
- Protocol design for multi-access broadcast channels



- Coordination of mobile robots
- Decentralized detection and target tracking

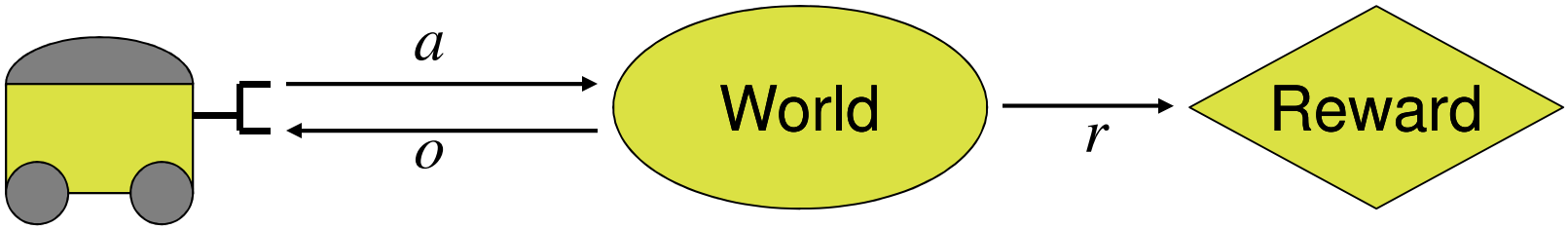
- Decentralized detection of hazardous weather events

Markov Decision Process (MDP)



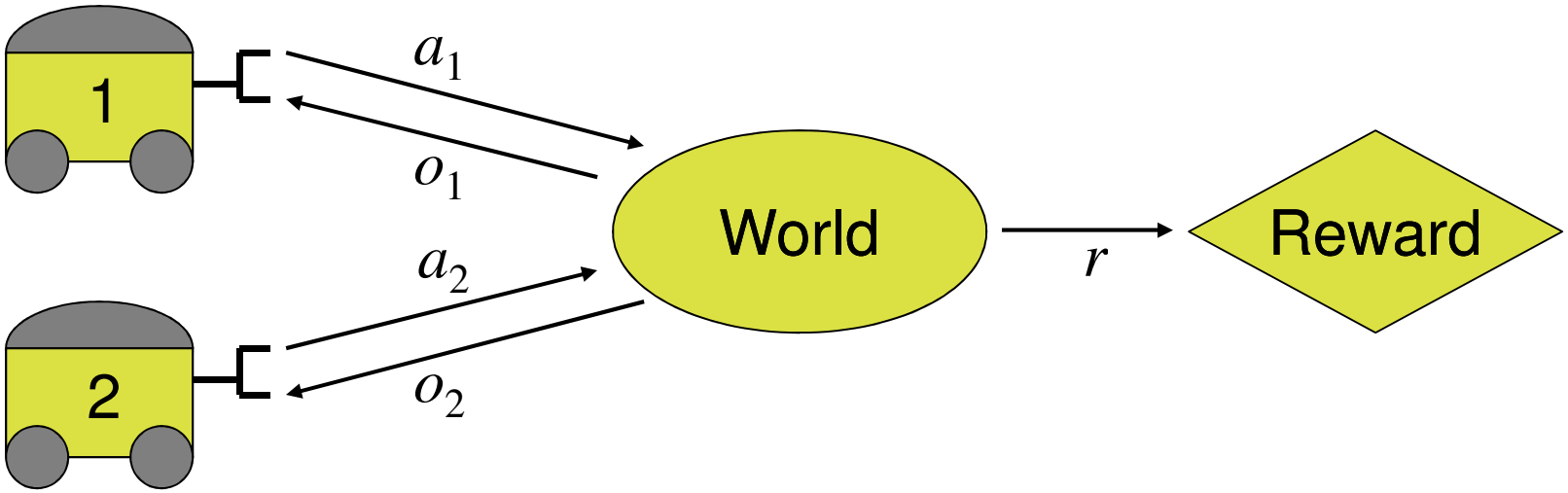
- Expressive model for stochastic planning
- Originated in operations research in the 1950s
- Adopted by the AI community as a framework for planning and learning under uncertainty
- Can be solved efficiently by DP algorithms and a range of search and abstraction methods
- Everything is an MDP – just keep adding states!

Partially Observable MDP



- Generalization formulated in the 1960s [Astrom 65]
- The agent receives noisy observations of the underlying world state
- Need to remember previous observations in order to act optimally
- More difficult, but there are DP algorithms [Smallwood & Sondik 73]

Decentralized POMDP



- Generalization of POMDP involving multiple cooperating decision makers, each receiving a different partial observation after a joint action is taken

DEC-POMDPs

Definition *A decentralized partially observable Markov decision process (DEC-POMDP) is a tuple* $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T \rangle$ *where*

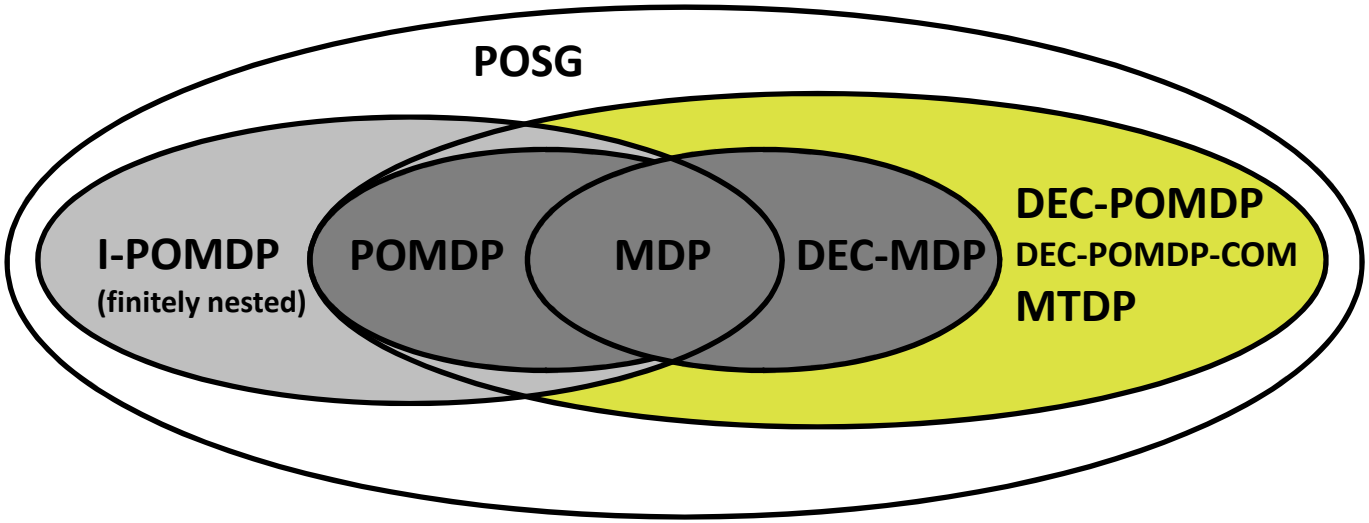
- I *is a finite set of agents indexed* $1, \ldots, n$.
- S *is a finite set of states, with distinguished initial state* s_0 *or belief state* b_0
- A_i *is a finite set of actions available to agent* i *and* $\vec{A} = \otimes_{i \in I} A_i$ *is the set of joint actions, where* $\vec{a} = \langle a_1, \ldots, a_n \rangle$ *denotes a joint action.*
- $P : S \times \vec{A} \to \Delta S$ *is a Markovian transition function.* $P(s'|s, \vec{a})$ *denotes the probability of a transition to state* s' *after taking joint action* $\vec{a}$ *in state* s.
- Ω_i *is a finite set of observations available to agent* i *and* $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ *is the set of joint observation, where* $\vec{o} = \langle o_1, \ldots, o_n \rangle$ *denotes a joint observation.*
- $O : \vec{A} \times S \to \Delta \vec{\Omega}$ *is an observation function.* $O(\vec{o}|\vec{a}, s')$ *denotes the probability of observing joint observation* $\vec{o}$ *given that joint action* $\vec{a}$ *was taken and led to state* s'. *Here* $s' \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.
- $R : \vec{A} \times S \to \Re$ *is a reward function.* $R(\vec{a}, s')$ *denotes the reward obtained after joint action* $\vec{a}$ *was taken and a state transition to* s' *occurred.*

Subclasses and Related Models

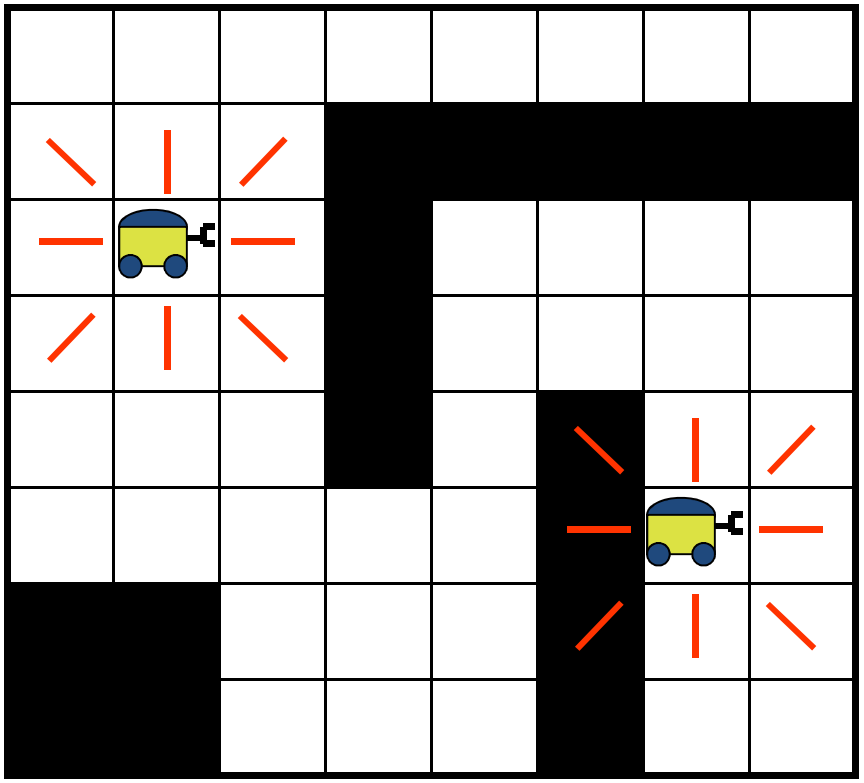
- ***Decentralized MDP*** (DEC-MDP): DEC-POMDP in which the combined observations of all the agents provide perfect information about the underlying world state
- ***Multiagent MDP*** (MMDP): DEC-MDP in which each agent has perfect information about the underlying state
- ***Partially-Observable Stochastic Game*** (POSG): Generalization of DEC-POMDP in which each agent can have a different objective function.
- ***Interactive POMDP*** (I-POMDP): A model in which each agent explicitly represents beliefs about the other agents and about the world state.

Relationship Between Models

Relationships among the various decision-theoretic models



Example: Mobile Robot Planning



States: grid cell pairs

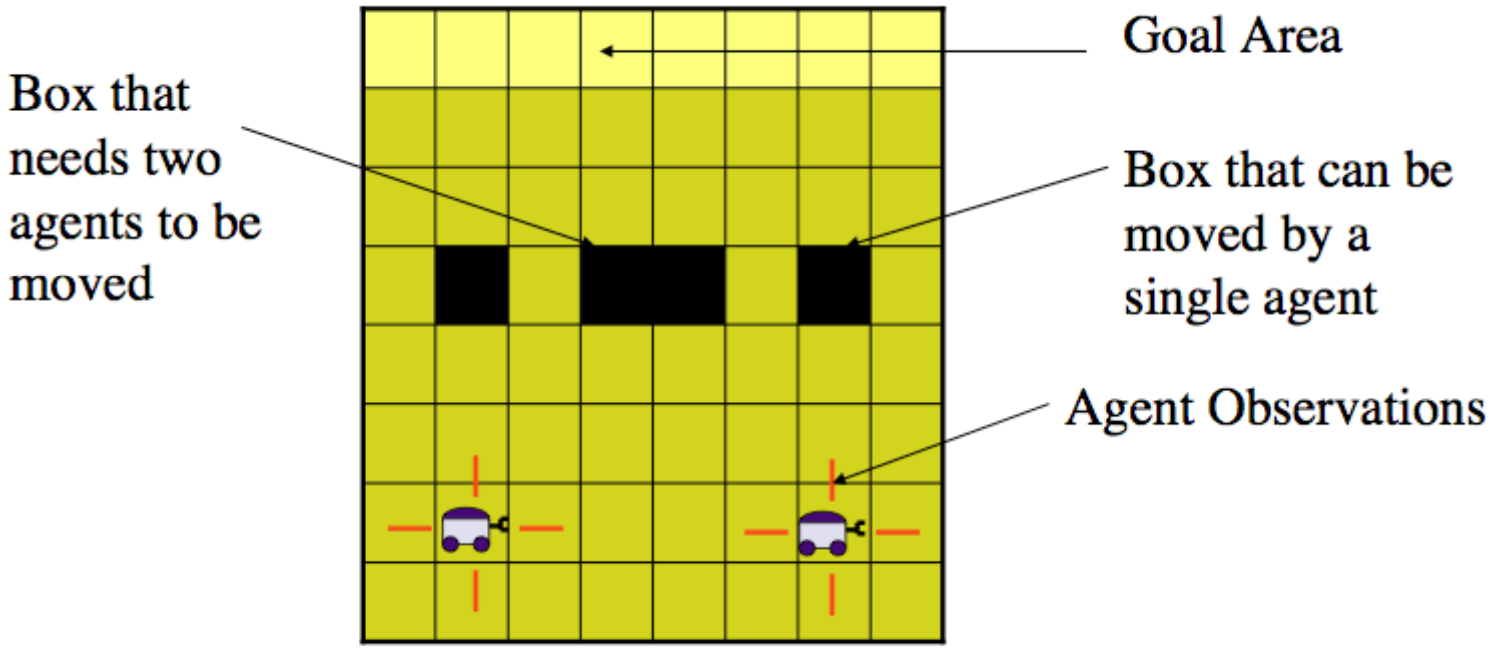
Actions: $\uparrow, \downarrow, \leftarrow, \rightarrow$

Transitions: noisy

Goal: meet quickly

Observations: red lines

Example: Cooperative Box-Pushing



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

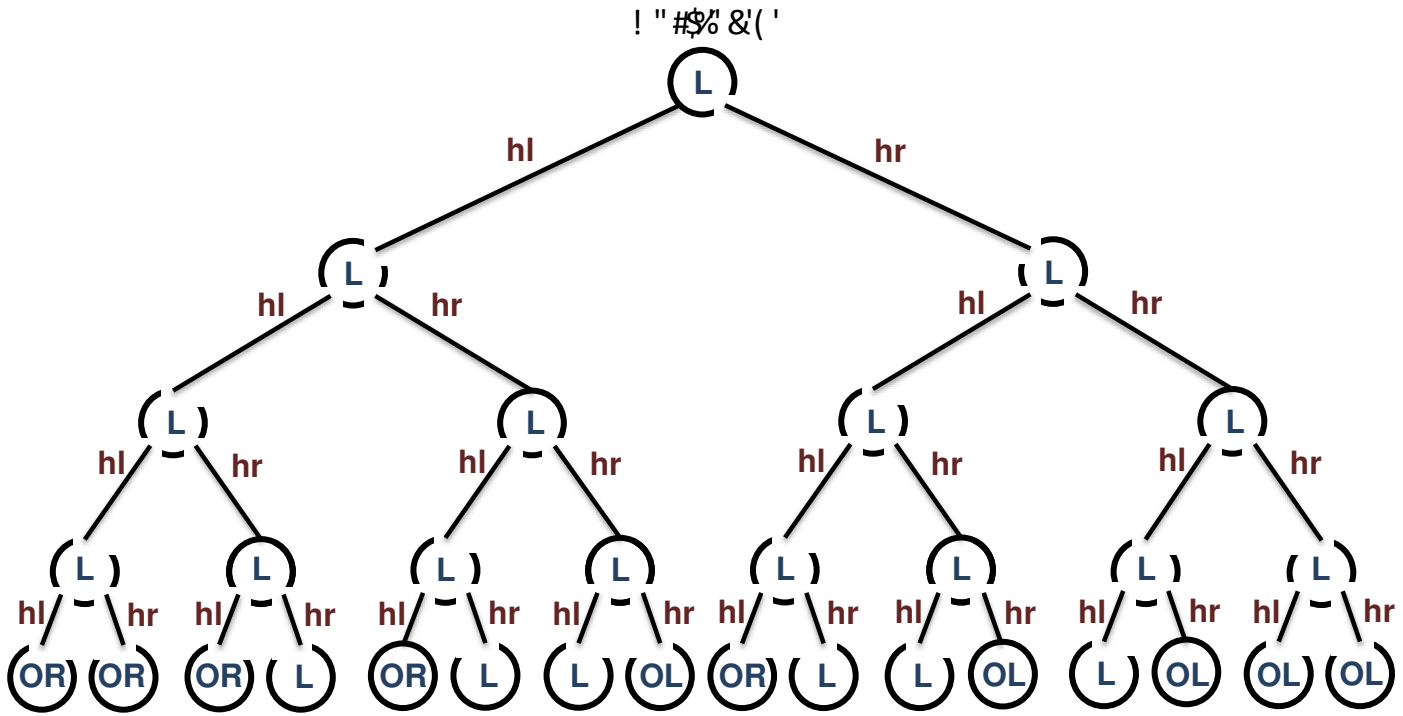
Example: Multiagent Tiger Problem

- A simple toy problem used for illustration with 2 agents, 2 states, 3 actions and 2 observations [Nair et al. 03]
- Two agents are situated in a room with two doors. Behind one door is a tiger and behind the other is a large treasure.
- Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action a larger positive reward or a smaller penalty is given to reward cooperation.
- Listening incurs a small cost and provide a noisy observation of which door the tiger is behind.

Solution Representation

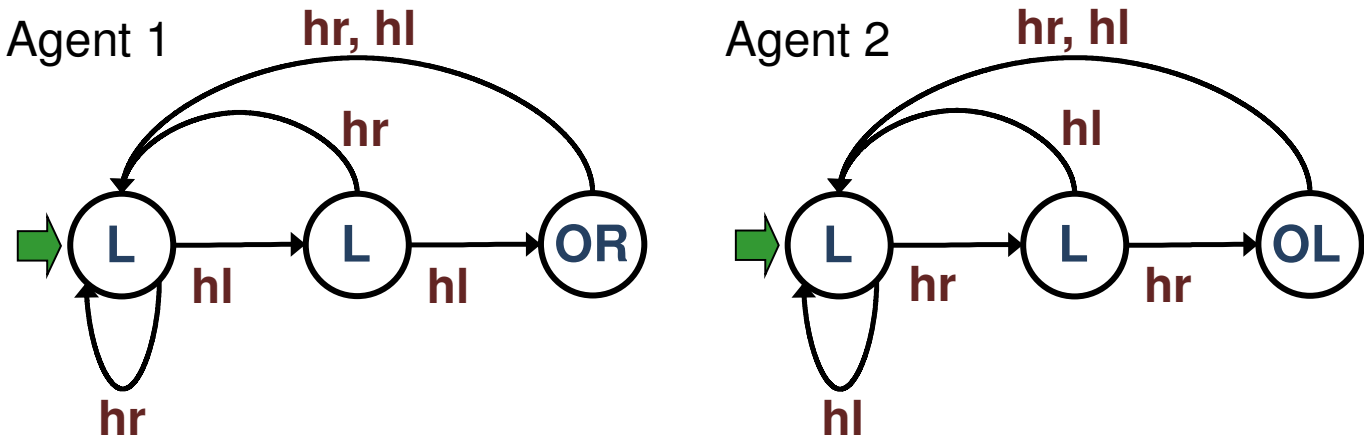
- Each agent's behavior is described by a **local policy** δ_i
- Policy can be represented as a mapping from
  - Local **observation sequences** to **actions**; or
  - Local **memory states** to **actions**
- Actions can be selected **deterministically** or **stochastically**
- Goal is to **maximize expected reward** over a finite horizon or discounted infinite horizon

Solutions as Policy Trees



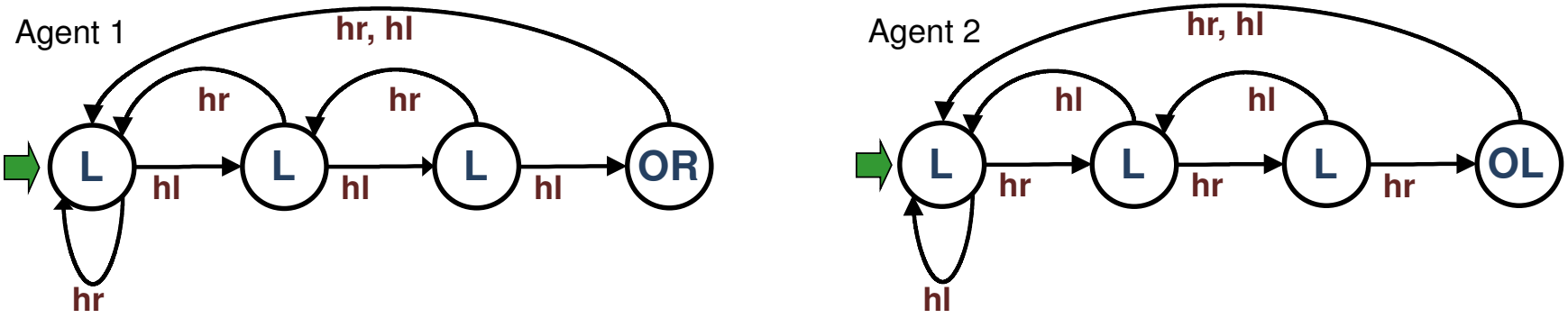
- Each node is labeled with an action and each edge with an observation that could be received
- Policy tree shown above is optimal for the multiagent tiger problem with horizon 5. (Same tree assigned to both agents)

Solutions as Finite-State Controllers



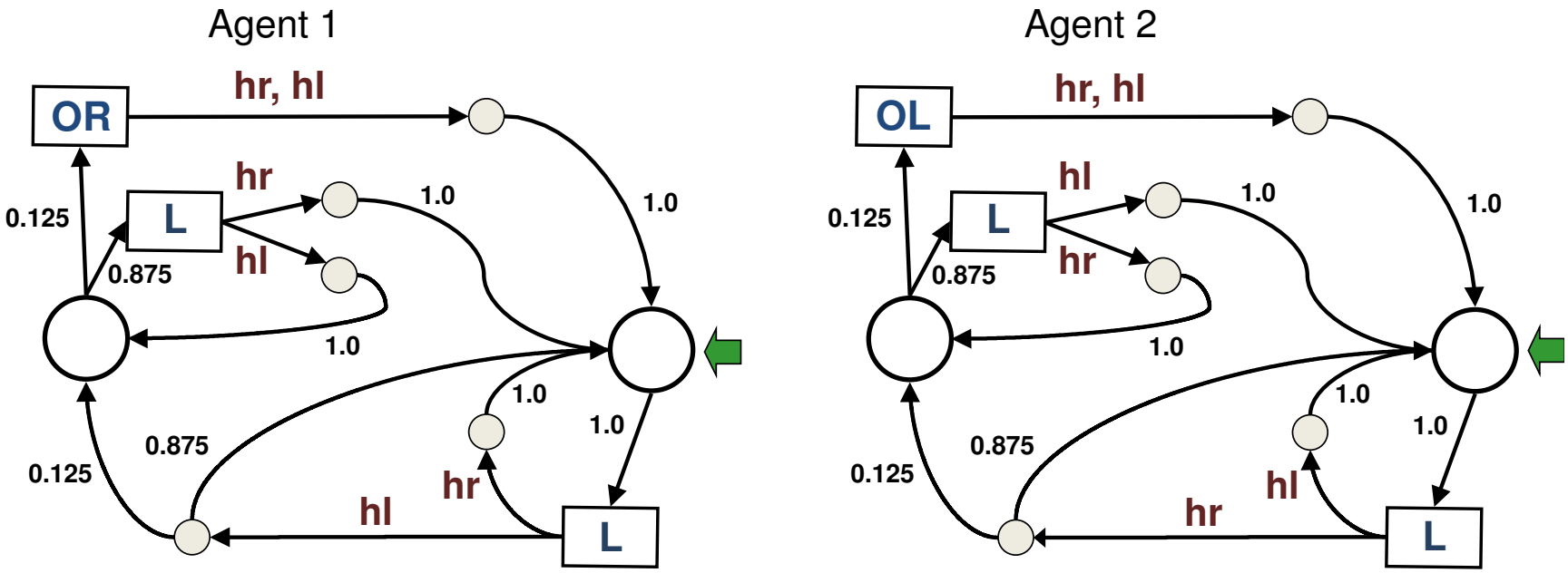
- Each controller state is labeled with an action and edges between states are labeled with observations.
- Shown above are optimal three-node deterministic controllers for the multiagent tiger problem.
- Green arrow designate the initial state of the controller.

Solutions as Finite-State Controllers



- Optimal four-node deterministic controllers for the multiagent tiger problem.
- The policies assigned to the agents are different.

Stochastic Controllers



- Stochastic two-node controllers for multiagent tiger.
- In each controller state, actions are selected stochastically; when an observation is obtained, the transition to a new state is also stochastic.

Evaluating Solutions

- For a finite-horizon problem with initial state s_o and T time steps, the value of a joint policy δ is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{T-1} R(\vec{a}_t, s_t) | s_0, \delta\right].$$

- For an infinite-horizon problem, with initial state s_o and discount factor γ in [0;1), the value of a joint policy δ is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(\vec{a}_t, s_t) | s_0, \delta\right].$$

Previous Complexity Results

Finite Horizon

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

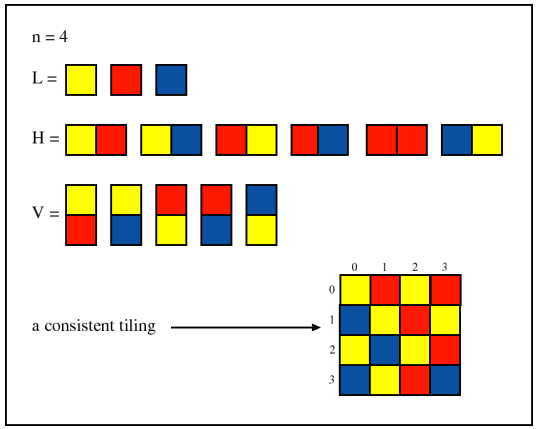
Infinite-Horizon Discounted

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

- A **static** version of the problem, where a **single** set of decisions is made in response to a set of observations, was shown to be NP-hard [Tsitsiklis and Athan, 1985]
- Bernstein et al. proved that two-agent finite-horizon DEC-POMDPs are **NEXP-hard** via a reduction to TILING
- But these are worst-case results! Are real-world problems easier?

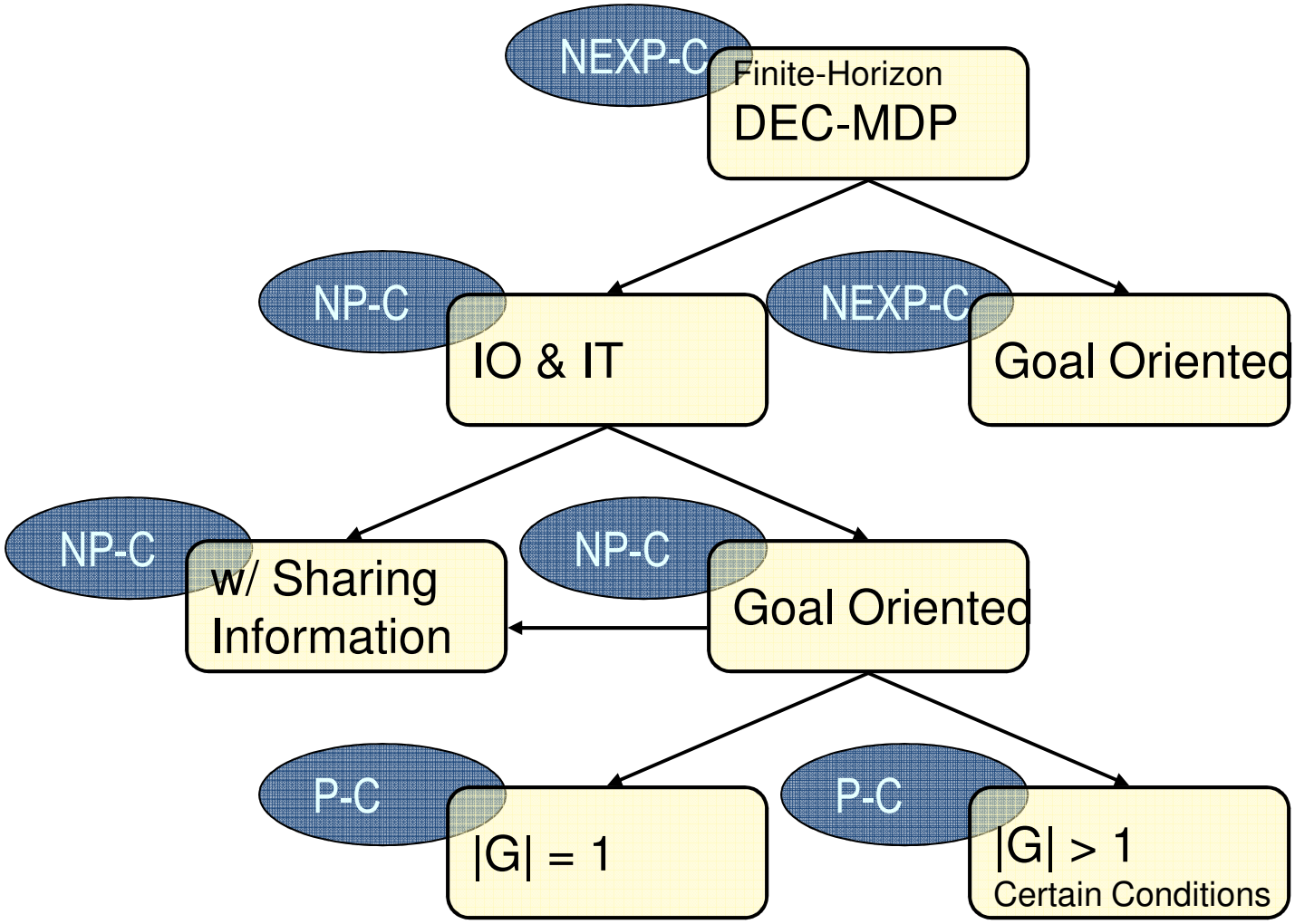


What Features of the Domain Affect the Complexity and How?

- 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 and structure of interaction
- Degree of information sharing and communication

Complexity of Sub-Classes

Goldman & Zilberstein, JAIR 2004



Solving Finite-Horizon DEC-POMDPs

Notation

- q_i denotes a policy tree and Q_i a set of policy trees for agent i.
- Q_{-i} denotes the sets of policy trees for all agents but agent i.
- A joint policy $\vec{q} = \langle q_1, q_2, \cdots, q_n \rangle$ is a vector of policy trees and $\vec{Q} = \langle Q_1, Q_2, \cdots, Q_n \rangle$ denotes the sets of joint policies.

Evaluating a joint policy

$$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V(s', \vec{q}_{\vec{o}})$$

where $\vec{a}$ are the actions at the roots of trees $\vec{q}$ and
$\vec{q}_{\vec{o}}$ are the subtrees of $\vec{q}$ after obtaining observations $\vec{o}$.

JESP: First DP Algorithm

Nair, Tambe, Yokoo, Pynadath & Marsella, IJCAI 2003

Algorithm 3: DP-JESP for DEC-POMDPs

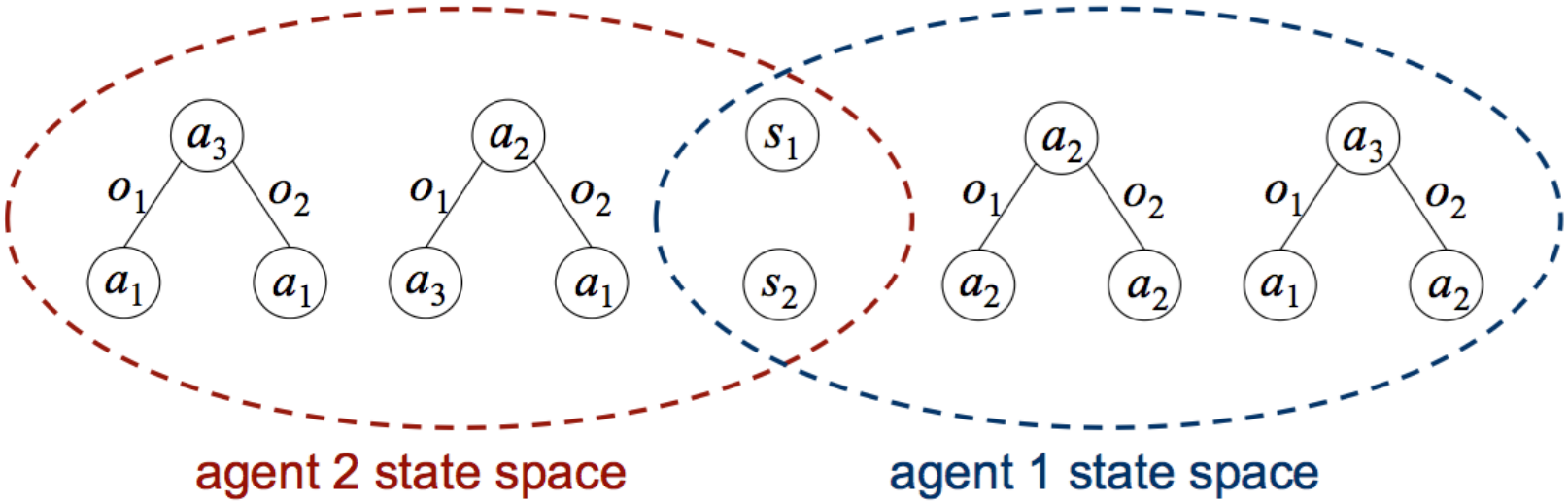
Generate a random joint policy ;
repeat
  foreach *agent i* **do**
    Fix the policies of all the agents except i ;
    for $t=T$ **downto** *1* **do** // `forwards`
      Generate a set of all possible belief state:
      $B^t(\cdot|B^{t+1}, q^t_{-i}, a_i, o_i), \forall a_i \in A_i, \forall o_i \in \Omega_i$;
    end
    for $t=1$ **to** T **do** // `backwards`
      foreach $b^t \in B^t$ **do**
        Compute the best value for $V^t(b^t, a_i)$;
      end
    end
    forall the *possible observation sequences* **do**
      for $t=T$ **downto** *1* **do** // `forwards`
        Update the belief state b^t given q^t_{-i} ;
        Select the best action according to $V^t(b^t, a_i)$;
      end
    end
  end
until *no improvement in the policies of all agents*;
return the current joint policy ;

- JESP: Joint Equilibrium-based Search for Policies
- Complexity: exponential
- Result: only locally optimal solutions

Is Exact DP Possible?

- The key to solving POMDPs is that they can be viewed as **belief-state** MDPs [Smallwood & Sondik 73]
- Not as clear how to define a belief-state MDP for a DEC-POMDP
- The first exact DP algorithm for finite-horizon DEC-POMDPs used the notion of a **generalized belief state**
- The algorithm also applies to competitive situations modeled as POSGs

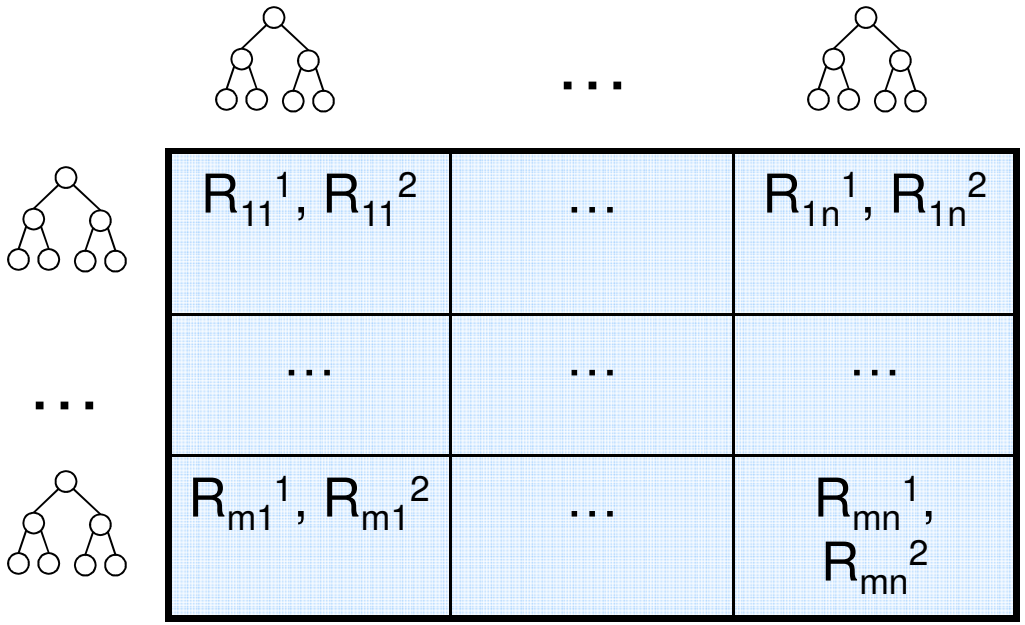
Generalized Belief State



A generalized belief state captures the uncertainty of one agent with respect to the state of the world as well as the policies of other agents.

Strategy Elimination

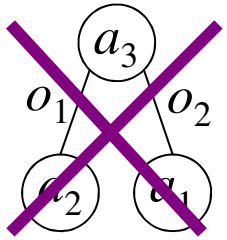
- Any finite-horizon DEC-POMDP can be converted to a **normal form game**
- But the number of strategies is **doubly exponential** in the horizon length!



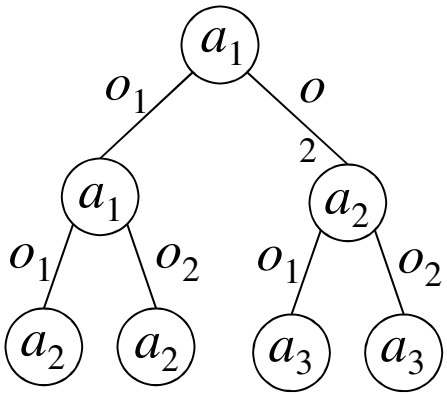
A Better Way to Do Elimination

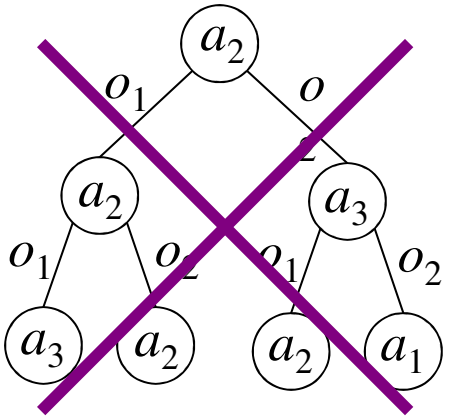
Hansen, Bernstein & Zilberstein, AAAI 2004

- 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)

Exact DP for DEC-POMDPs

Hansen, Bernstein & Zilberstein, AAAI 2004

Algorithm 4: Exact DP for DEC-POMDPs

Initialize all depth-1 policy trees ;
for $t=1$ **to** T **do** // backwards
  Perform full backup on $\vec{Q}^t$;
  Evaluate policies in $\vec{Q}^{t+1}$;
  Prune dominated policies in $\vec{Q}^{t+1}$;
end
return the best joint policy in $\vec{Q}^T$ for b^0 ;

- Algorithm is complete & optimal
- Complexity is double exponential

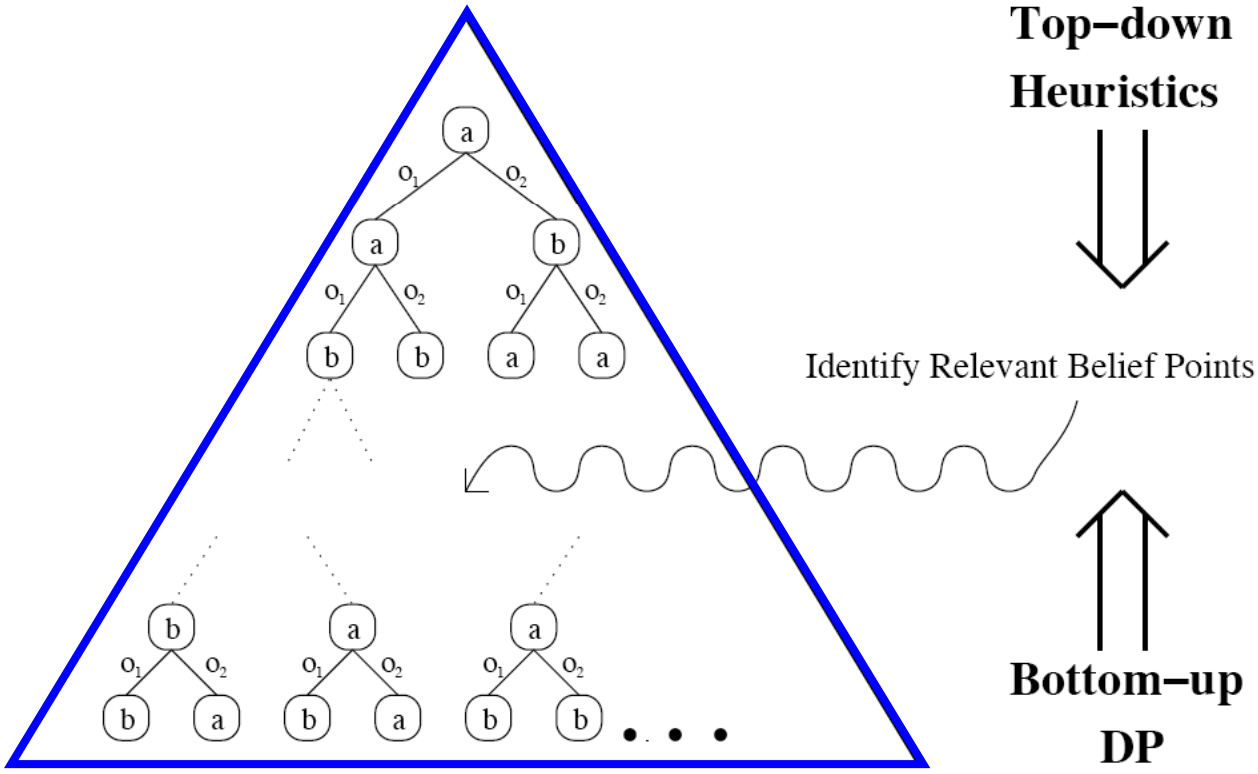
- **Theorem:** *DP performs iterated elimination of dominated strategies in the normal form of the POSG.*
- **Corollary:** *DP can be used to find an optimal joint policy in a DEC-POMDP.*

Memory-Bounded DP (MBDP)

Seuken & Zilberstein, IJCAI 2007

- Combining two approaches:
  - The DP algorithm that operates **bottom-up**
  - Heuristic search that operates **top-down**
- The DP step can only eliminate a policy tree if it is dominated for *every* belief state
- But, only a small subset of the belief space is actually reachable
- Furthermore, the combined approach allows the algorithm to focus on a small subset of joint policies that appear best

Memory-Bounded DP Cont.



The MBDP Algorithm

Algorithm 5: Memory-Bounded DP for DEC-POMDPs

Initialize all depth-1 policy trees ;
for t=*1* **to** T **do** // backwards
    Perform full backup on $\vec{Q}^t$;
    Evaluate policies in $\vec{Q}^{t+1}$;
    for k=*1* **to** *maxTrees* **do**
        Select a heuristic h from the heuristic portfolio ;
        Generate a state distribution $b \in \Delta(S)$ using h ;
        Select the best joint policy $\vec{q}^{\,t+1}$ in $\vec{Q}^{t+1}$ for b ;
    end
    Prune all the policies except the selected ones ;
end
return the best joint policy in $\vec{Q}^T$ for b^0 ;

Generating “Good” Belief States

- **MDP Heuristic --** Obtained by solving the corresponding fully-observable multiagent MDP
- **Infinite-Horizon Heuristic --** Obtained by solving the corresponding infinite-horizon DEC-POMDP
- **Random Policy Heuristic --** Could augment another heuristic by adding random exploration
- **Heuristic Portfolio --** Maintain a set of belief states generated by a set of different heuristics
- **Recursive MBDP**

Performance of MBDP

	MABC Problem				Tiger Problem				
Horizon	Optimal	PBDP	Random	**MBDP**	Optimal	JESP	PBDP	Random	**MBDP**
2	2.00	2.00	1	**2.00**	-4.00	-4.00	-4.00	-92.44	**-4.00**
3	2.99	2.99	1.49	**2.99**	5.19	-6.00	5.19	-138.67	**5.19**
4	3.89	3.89	1.99	**3.89**	4.80	?	4.80	-184.89	**4.80**
5	-	4.70	2.47	**4.79**	-	?	-	-231.11	**5.38**
6	-	5.69	2.96	**5.69**	-	?	-	-277.33	**9.91**
7	-	6.59	3.45	**6.59**	-	?	-	-323.56	**9.67**
8	-	7.40	3.93	**7.49**	-	-	-	-369.78	**9.42**
9	-	-	4.41	**8.39**	-	-	-	-416.00	**12.57**
10	-	-	4.90	**9.29**	-	-	-	-462.22	**13.49**
100	-	-	48.39	**90.29**	-	-	-	-4,622.22	**93.24**
1,000	-	-	483.90	**900.29**	-	-	-	-46,222.22	**819.01**
10,000	-	-	4,832.37	**9,000.29**	-	-	-	-462,222.22	**7930.68**
100,000	-	-	48,323.09	**90,000.29**	-	-	-	-4,622,222.22	**78252.18**

MBDP Parameter Tuning

The best parameter settings and solution values for the tiger problem with horizon 20 for given time limits.

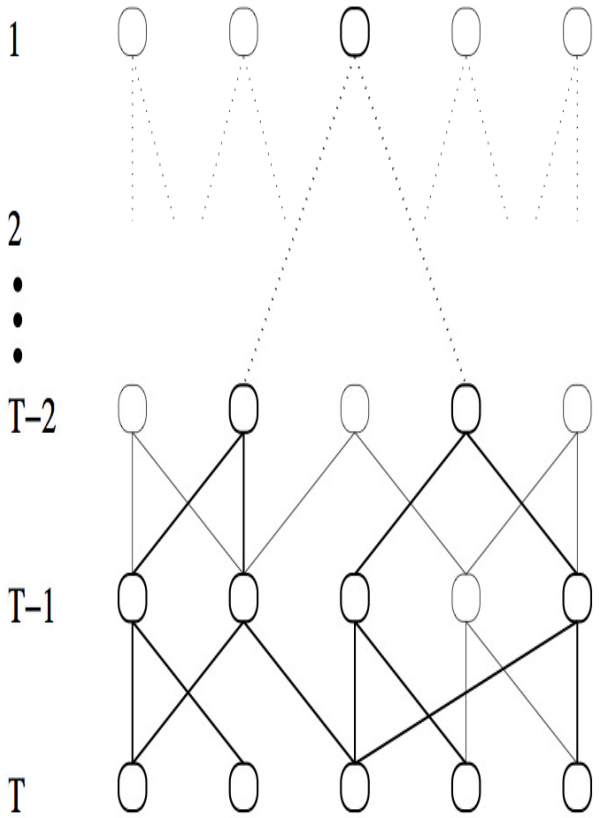
Time Limit (s)	MaxTrees	Recursion Depth	Value
1	7	1	16.37
2	7	2	20.35
3	7	4	22.34
4	8	3	23.67
5	8	3	23.67
10	11	1	24.71
20	10	5	26.98
30	12	3	27.16
60	9	27	28.75
120	13	8	29.02

MBDP Successors

- **Improved MBDP (IMBDP)**
  [Seuken and Zilberstein, UAI 2007]
- **MBDP with Observation Compression (MBDP-OC)**
  [Carlin and Zilberstein, AAMAS 2008]
- **Point Based Incremental Pruning (PBIP)**
  [Dibangoye, Mouaddib, and Chaib-draa, AAMAS 2009]
- **PBIP with Incremental Policy Generation (PBIP-IPG)**
  [Amato, Dibagoye, Zilberstein, AAAI 2009]
- **Constraint-Based Dynamic Programming (CBDP)**
  [Kumar and Zilberstein, AAMAS 2009]
- **Point-Based Backup for Decentralized POMDPs**
  [Kumar and Zilberstein, AAMAS 2010]
- **Point-Based Policy Generation (PBPG)**
  [Wu, Zilberstein, and Chen, AAMAS 2010]

Why Does MBDP Work?

- Perform search in a **reduced** policy space
- Exact algorithm performs only lossless pruning
- Approximate algorithms rely on more aggressive pruning
- MBDP represents an exponential size policy with **linear space** $O(maxTrees \times T)$
- Resulting policy is an **acyclic finite-state controller**.



Solving Infinite-Horizon DEC-POMDPs

- Unclear how to define a compact **belief-state** without fixing the policies of 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 (deterministic/stochastic) **finite-state controllers** and defining a set of **controller transformations** that guarantee improvement & convergence

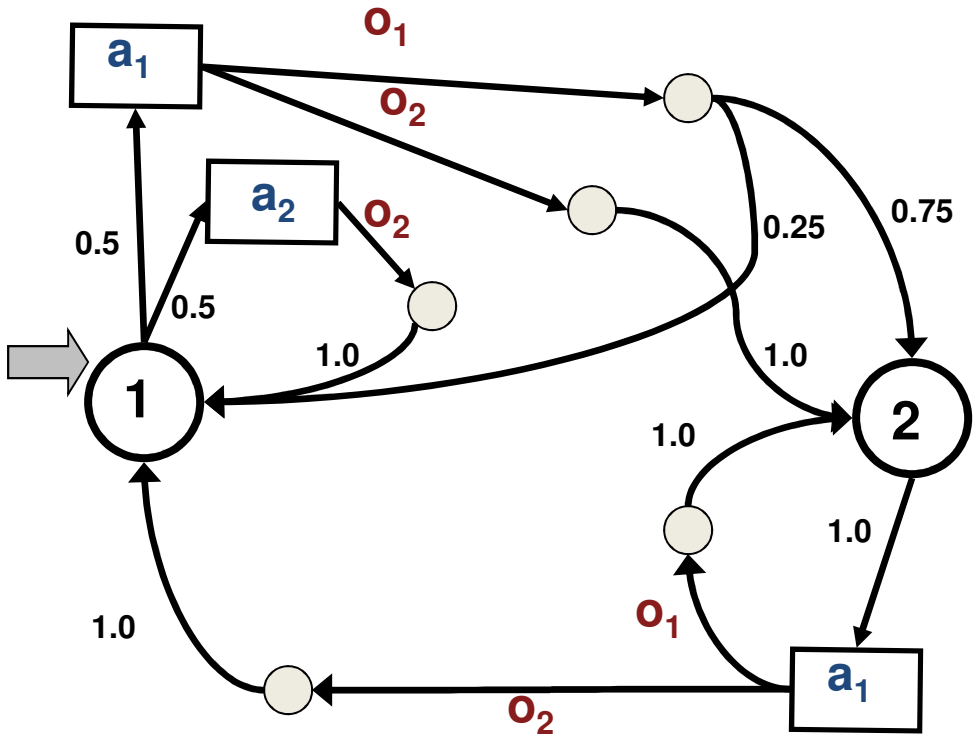
Policies as Controllers

- Finite state controller represents each policy
  - Fixed memory
  - Randomness used to offset memory limitations
  - Action selection, $\mu_i : Q_i \rightarrow \Delta A_i$
  - Transitions, $\tau_i : Q_i \times A_i \times O_i \rightarrow \Delta Q_i$
- 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
    - $P(a_i \mid q_i)$
    - $P(q_i' \mid q_i, o_i)$

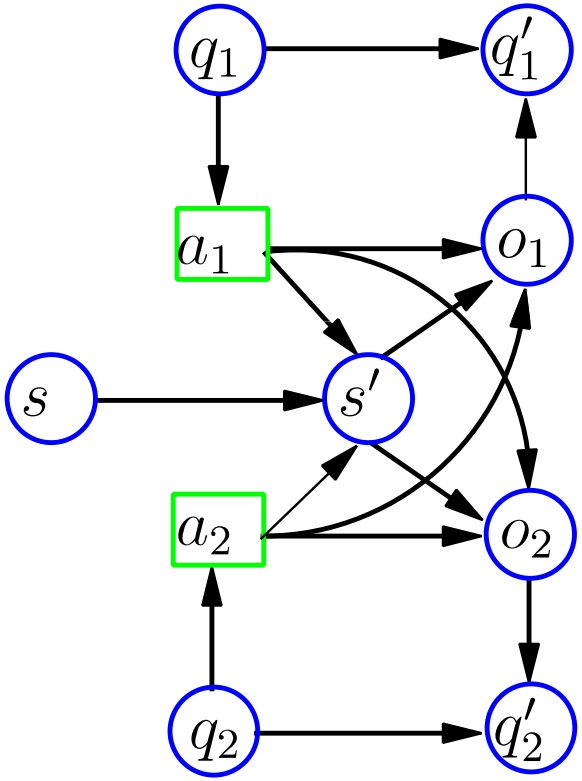


Finding Optimal Controllers

- 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 BSF or B&B
- Stochastic controllers – continuous optimization problem
- **Key question**: how to best use a **limited amount of memory** to optimize value?

Independent Joint Controllers

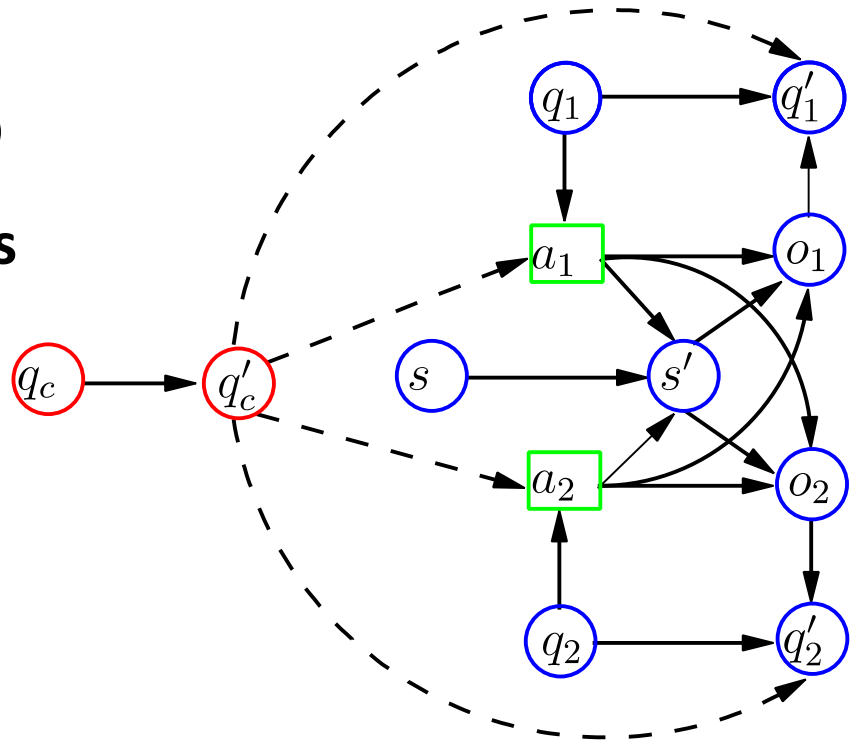
- Local controller for agent i is defined by conditional distribution $P(a_i, q_i \mid q_i, o_i)$
- Independent **joint controller** is expressed by: $\Pi_i P(a_i, q_i \mid q_i, o_i)$
- Can be represented as a **dynamic Bayes net**



Correlated Joint Controllers

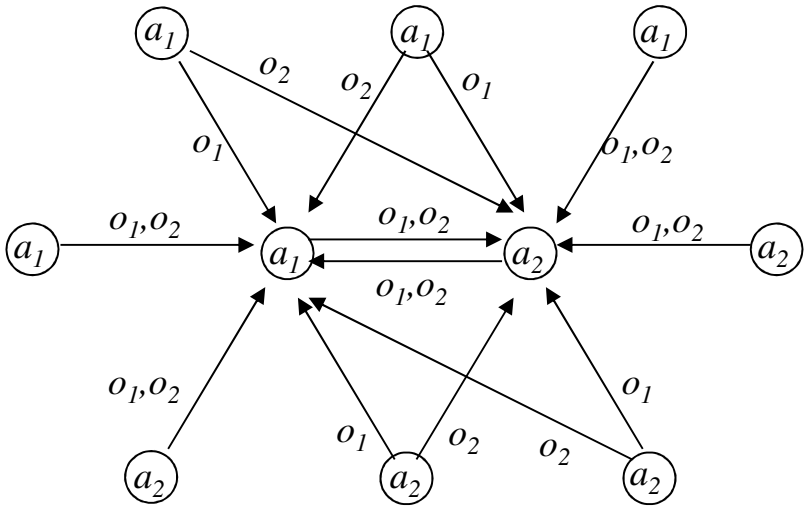
Bernstein, Hansen & Zilberstein, IJCAI 2005, JAIR 2009

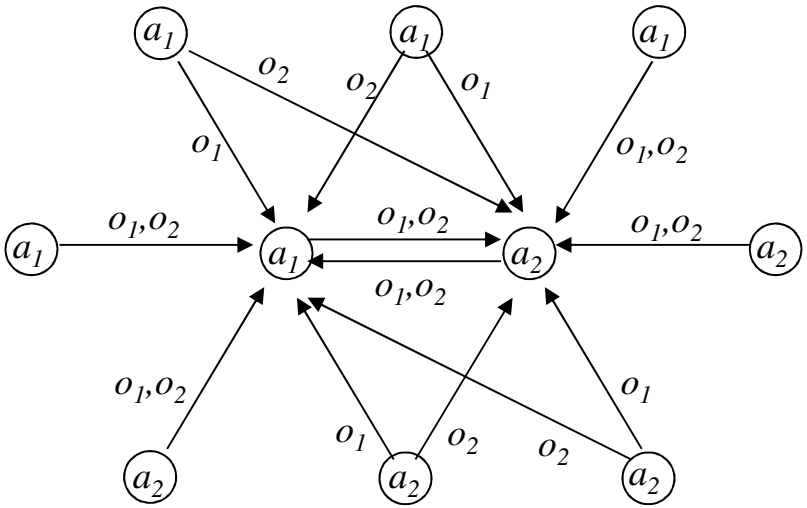
- A **correlation device**, $[Q_c, \psi]$, is a set of nodes and a stochastic state transition function
- Joint controller:
  $\sum_{q_c} P(q_c | q_c') \prod_i P(a_i, q_i' | q_i, o_i, q_c')$
- A **shared source of randomness** affecting decisions and memory state update
- Random bits for the correlation device can be determined prior to execution time



Exhaustive Backups

- Add a node for every possible action and deterministic transition rule





- Repeated backups converge to optimality, but lead to very large controllers

Value-Preserving Transformations

- A **value-preserving transformation** changes the joint controller without sacrificing value
- Formally, there must exist mappings

  $f_i : Q_i \rightarrow \Delta R_i$ for each agent i and $f_c : Q_c \rightarrow \Delta R_c$ such that

$$V(s,\vec{q},q_c) \leq \sum_{\vec{r}} P(\vec{r} \mid \vec{q}) \sum_{r_c} P(r_c \mid q_c) V(s,\vec{r},r_c)$$

  for all $s \in S, \vec{q} \in \vec{Q}$, and $q_c \in Q_c$

Bounded Policy Iteration Algorithm

Bernstein, Hansen & Zilberstein, IJCAI 2005, JAIR 2009

Algorithm 6: Policy Iteration for Infinite-Horizon DEC-POMDPs

input : DEC-POMDP, correlated joint controller, convergence parameter ε
output: A correlated joint controller that is ε-optimal for all states.
begin
  $t \leftarrow 0$;
  while $\gamma^{t+1} \cdot |R_{max}|/(1-\gamma) > \varepsilon$ **do**
    $t \leftarrow t + 1$;
    Evaluate correlated joint controller by solving a system of linear equations;
    Perform an exhaustive backup to add nodes to the local controllers;
    Perform value-preserving transformations on the controller;
  end
  return correlated joint controller;
end

Theorem: *For any ε, bounded policy iteration returns a joint controller that is ε-optimal for all initial states in a finite number of iterations.*

Useful Transformations

- Controller reductions
  - Shrink the controller without sacrificing value
- Bounded dynamic programming updates
  - Increase value while keeping the size fixed
- Both can be done using polynomial-size linear programs
- Generalize ideas from POMDP literature, particularly the BPI algorithm [Poupart & Boutilier 03]

Decentralized BPI Summary

- DEC-BPI finds better and much more compact solutions than exhaustive backups
- A larger correlation device tends to lead to higher values on average
- Larger local controllers tend to yield higher average values *up to a point*
- But, bounded DP is limited by improving one controller at a time
- Linear program (one-step lookahead) results in local optimality and tends to “get stuck”

Nonlinear Optimization Approach

Amato, Bernstein & Zilberstein, UAI 2007, JAAMAS 2010

- Idea: Model the problem as a non-linear program (NLP)
- Consider node values (and FSC parameters) as variables
- NLP can take advantage of an initial state distribution
- Perform improvement and evaluation all in one step
- Additional constraints maintain valid values
- Notation:

For each agent i:

- Variable $x(q_i, a_i)$ represents $P(a_i|q_i)$
- Variable $y(q_i, a_i, o_i, q_i')$ represents $P(q_i'|q_i, a_i, o_i)$
- Variable $z(\vec{q}, s)$ represents $V(\vec{q}, s)$ where $\vec{q}^0$ is the initial node

The NLP Approach

For variables of each agent i: $x(q_i, a_i)$, $y(q_i, a_i, o_i, q_i')$ and $z(\vec{q}, s)$

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

The Bellman constraints:

$\forall \vec{q}, s \; z(\vec{q}, s) =$

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

And probability constraints for each agent i:

$$\forall q_i \sum_{a_i} x(q_i, a_i) = 1, \quad \forall q_i, o_i, a_i \sum_{q_i'} y(q_i, a_i, o_i, q_i') = 1$$

$$\forall q_i, a_i \; x(q_i, a_i) \geq 0, \quad \forall q_i, o_i, a_i \; y(q_i, a_i, o_i, q_i') \geq 0$$

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

Comparison of NLP & DEC-BPI

Amato, Bernstein & Zilberstein, UAI 2007, JAAMAS 2010

- 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

NLP vs. DEC-BPI for Box Pushing

Amato, Bernstein & Zilberstein, JAAMAS 2010

Values and running times (in seconds) for each controller size using NLP methods and DEC-BPI with and without a 2 node correlation device and BFS. An "x" indicates that the approach was not able to solve the problem.

Value

	NLO	NLO corr	NLO fix	NLO fix corr	DEC-BPI	DEC-BPI corr	BFS
1	-1.580	6.267	n/a	n/a	-10.367	-10.012	-2
2	31.971	39.825	-6.252	-10.865	3.293	4.486	x
3	46.282	50.834	5.097	5.300	9.442	12.504	x
4	50.640	x	18.780	25.590	7.894	x	x
5	x	x	53.132	53.132	14.762	x	x
6	x	x	73.247	x	x	x	x
7	x	x	80.469	x	x	x	x

Time

	NLO	NLO corr	NLO fix	NLO fix corr	DEC-BPI	DEC-BPI corr	BFS
1	20	21	n/a	n/a	26	87	1696
2	115	314	18	21	579	3138	x
3	683	1948	27	37	4094	15041	x
4	5176	x	44	108	11324	x	x
5	x	x	92	12189	27492	x	x
6	x	x	143	x	x	x	x
7	x	x	256	x	x	x	x

NLP Approach Summary

- The NLP defines the optimal fixed-size stochastic controller
- Approach shows consistent improvement over DEC-BPI using an off-the-shelf locally optimal solver
- A small correlation device can have significant benefits
- Better performance may be obtained by exploiting the structure of the NLP

Multiagent Execution

- In the simplest case where the model used for plan/control decision-making is faithful to the actual environment, agents simply follow their decisions until done.
- More generally, models are incomplete, and so the agents can face situations that they had not anticipated.
- Difficult enough in single-agent case; harder in multiagent case, where some agents' responses to unexpected situations can trigger other agents to encounter unexpected situations, leading to an unfortunate chain reaction.

Multiagent Plan Monitoring

- In the single-agent case, an agent can monitor its state and compare its state with what its plan anticipated to detect a deviation.
- In the multiagent case, it is possible that each agent's local state is consistent with one of its expected trajectories, and yet the global state might have deviated from the joint plan.
- Hence, detecting plan deviations is a distributed problem solving task, where agents might need to share partial, tentative hypotheses about the global state to determine when the global state has veered from expectations.

Multiagent Plan Recovery

- In the single-agent case, recovering from a deviation involves repairing the plan to get it back on course, or replanning from the current state.
- In the multiagent case, both of these strategies are possible as well. Unfortunately, repairing and replanning locally can introduce new coordination flaws into the joint plan, and thus can trigger a chain reaction of plan coordination.
- Further, deviations due to incorrect models mean that revised plans are prone to deviations themselves, unless models are updated based on experience. This raises issues in multiagent learning, and the danger of non-stationarity in simultaneous learning.

Continuous Multiagent Planning

- In some domains, deviations might be so prevalent that, if agents need to converge on fully-coordinated joint plans before continuing, they might never make meaningful progress.
- For more cognitive tasks, such as interpretation tasks (tracking vehicles in distributed sensor nets), it might be acceptable for agents to pursue plans that are not fully coordinated:
  - The consequences of miscoordination are not catastrophic.
- The idea is that agents can make local repairs to their plans, potentially anticipating repairs that other agents will make to their plans, and pursue the repaired plans without waiting to converge on fully-coordinated plans.

PGP: Partial Global Planning

- Coordination over abstract actions: This makes it faster (so agents reestablish full coordination sooner) and more robust (deviations in detailed plans might cancel each other out such that abstract plans remain coordinated.
- Decentralized coordination: While many of the coordination approaches in this chapter involve some decentralized processing, most ultimately require a single entity to make and impose coordination decisions/constraints. By not insisting on provably-coordinated plans at any given time, PGP is fully decentralized.
- Multiagent reasoning: To achieve decentralization, each agent reasons not only about its own plans, but about others' plans. In essence, each hypothesizes how the joint plan will change, and will enact its own changes in expectation that others will do the same.

PGP: Partial Global Planning (2)

- Communication planning: By examining the evolving plans it ascribes to others, an agent makes more informed decisions about what partial results to share to improve collective performance.
- Asynchronous responsiveness: Each agent pursues a plan based on its current best guess as to the global situation, permitting faster responses to emergent circumstances.
- Dampened responsiveness: Agents using PGP are endowed with a rudimentary understanding that coordination incurs cost and delay, and thus there might be some deviations from expectations that are sufficiently minor that correcting for them will actually degrade performance worse than proceeding with their existing plans.

Conclusions

- Multiagent planning and control has been addressed using various assumptions about degree of coupling in agents' activities, degree of mutual awareness possible, degree to which local plans are hard to formulate relative to resolving coordination problems, and the degree to which agents' environments are uncertain and non-deterministic.
- Multiagent planning/control has been a topic of research in the multiagent systems community from the community's inception.
- Multiagent planning/control under classical assumptions has drawn on a variety of concepts including social laws, organizations, contracting, state-space search, and hierarchical plan decomposition.
- In non-classical settings, contemporary techniques in multiagent planning/control draw heavily on mathematical programming, Markov decision processes, and Bayesian networks.