Autonomous Agents and Multiagent Systems

Multiagent planning

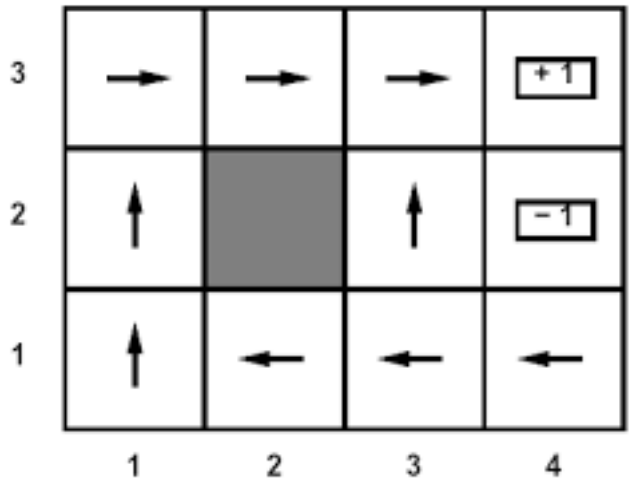
Francesco Amigoni

Plan

Informally, a plan is a sequence of actions that achieve a purpose represented:

- explicitly
- implicitly

```
 1: (INTEGRATECOMPONENT C5)
 2: (TESTCOMPONENT C5)
 3: (INTEGRATECOMPONENT C3)
 4: (TESTCOMPONENT C3)
 5: (INTEGRATECOMPONENT C4)
 6: (TESTCOMPONENT C4)
 7: (INTEGRATECOMPONENT C2)
 8: (TESTCOMPONENT C2)
 9: (INTEGRATECOMPONENT C1)
10: (TESTCOMPONENT C1)
```



Many of the following slides are taken from:

- the “lecture slides provided by authors”, Chapter 11, available at:
  http://www.the-mas-book.info/index.html
- the “AAAI 2019 Tutorial on Multi-Agent Path Finding: Models, Solvers, and Systems” by Roman Barták, Philipp Obermeier, Torsten Schaub, Tran Cao Son, and Roni Stern, available at:
  http://mapf.info
- the “CTS2012 Tutorial on Models of Coordination in Multiagent Decision Making” by Chris Amato, available at:
  http://masplan.org/tutorials

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.

Real-world examples

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

Real-world examples

ESA concurrent design

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

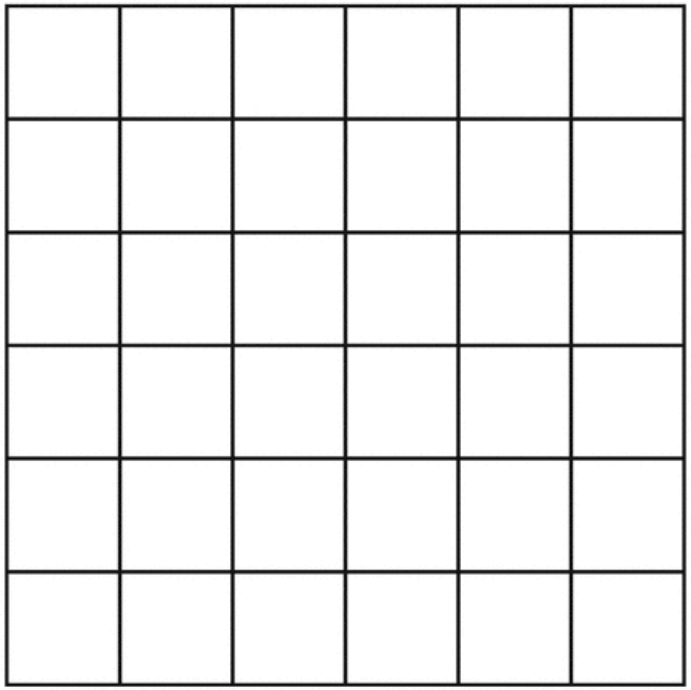
Social Laws

- 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

Collision avoidance

Multiple agents move in a grid environment in order to pick up and deliver objects

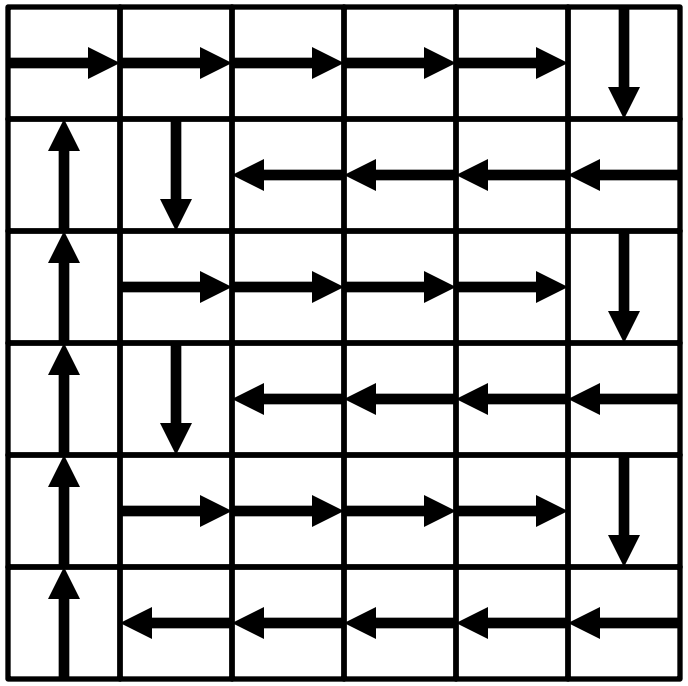
Agents should not collide with each other



Social Laws for Collision Prevention

- Never enter a location that is occupied.
  - Does not account for simultaneous movement, when more than one robot enters the same (previously) unoccupied location.
- 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 collision.
- But distance traveled will generally be larger than the minimum necessary.

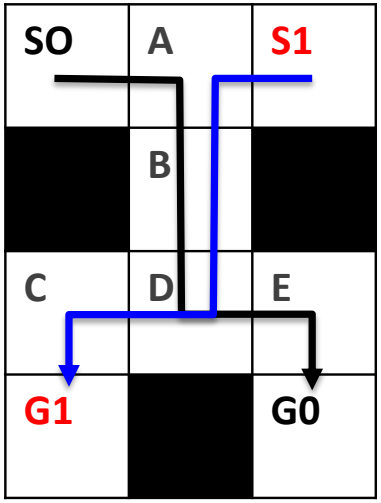
Conventions

- 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.

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.

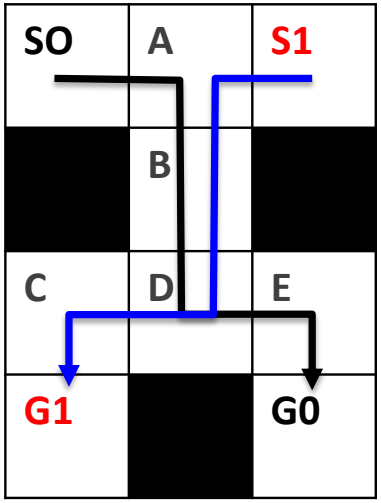
M (Wagner & Choset '11,'14)*

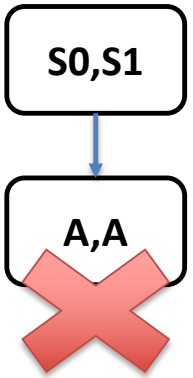


M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

M (Wagner & Choset '11,'14)*

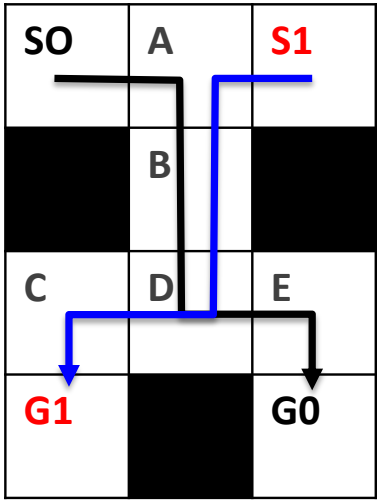


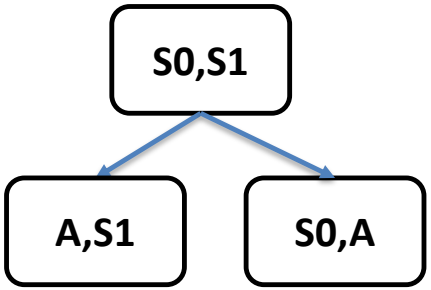


M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

M (Wagner & Choset '11,'14)*

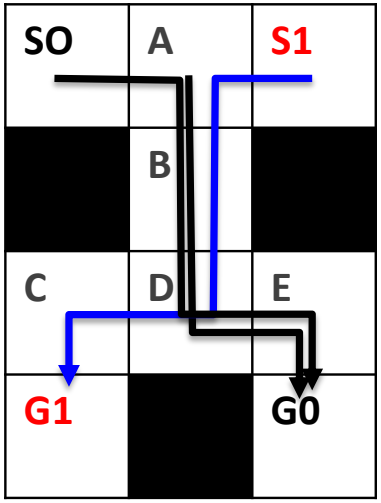


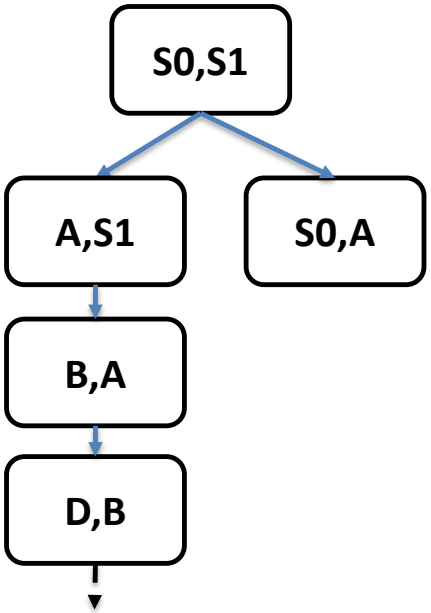


M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions

Recursive M (Wagner & Choset '11,'14)*





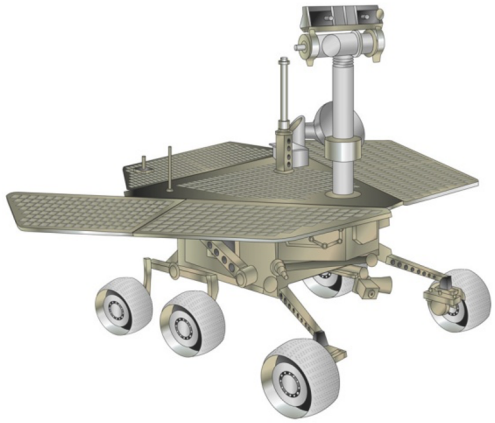
Recursive M*

1. Find optimal path for each agent individually
2. Start the search. **Generate only nodes on optimal paths**
3. If **conflict occurs – backtrack** and consider all ignored actions
   - Apply M* recursively after backtracking

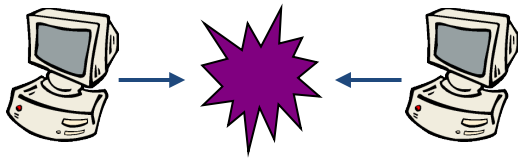
Decision-Theoretic MA Planning

- 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**

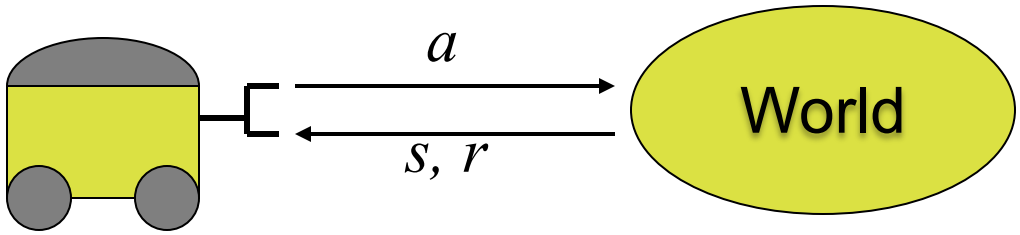
Applications



- 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

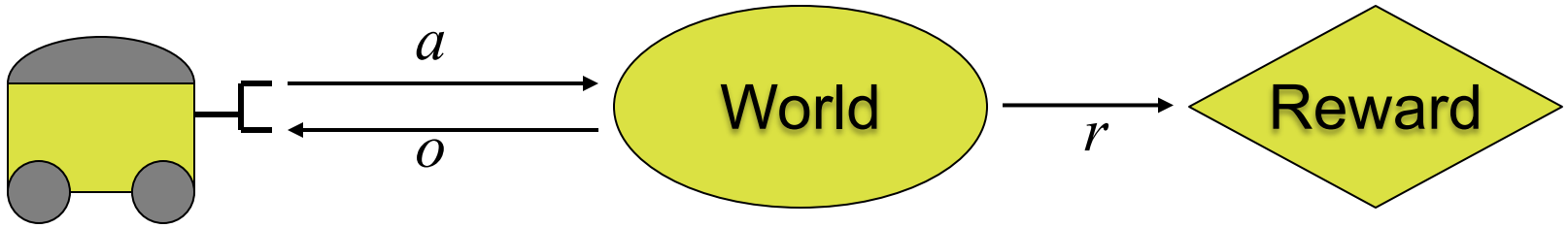


Markov Decision Process (MDP)



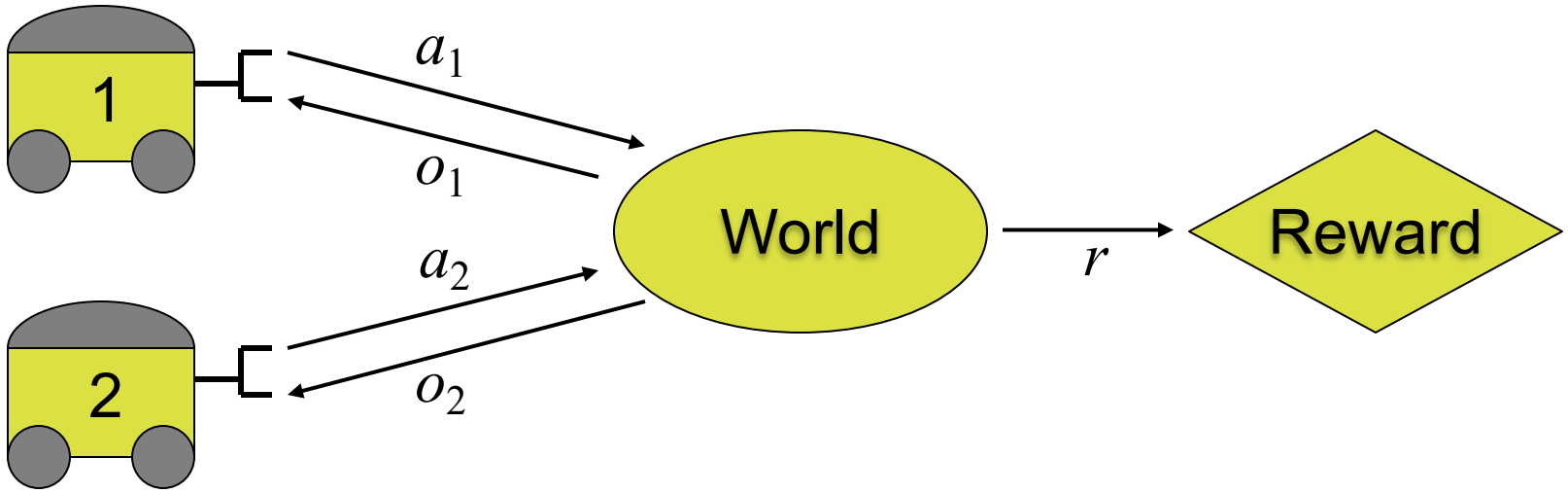
- Expressive model for stochastic planning
- Originated in operations research in the 1950s
- Adopted by the AI community as a framework for planning and learning under uncertainty
- Can be solved efficiently by DP algorithms and a range of search and abstraction methods

Partially Observable MDP



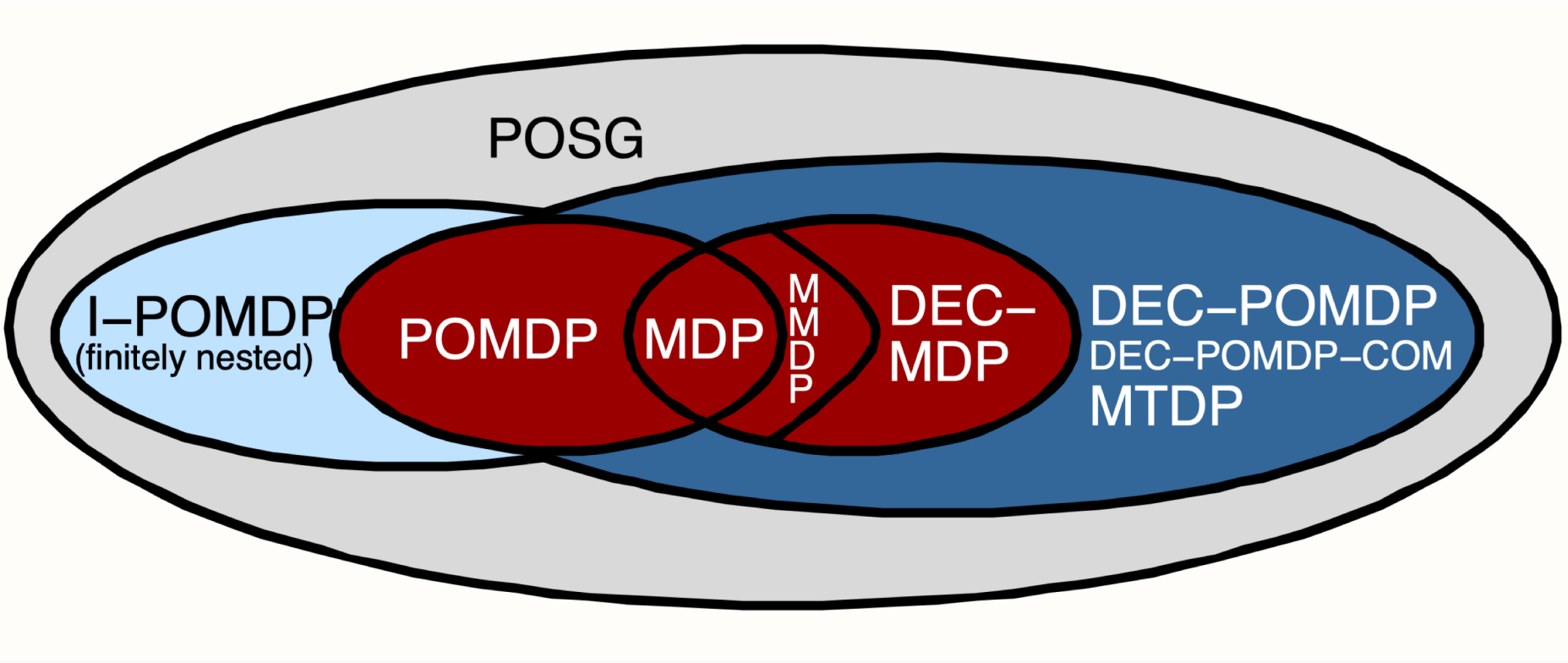
- Generalization formulated in the 1960s [Astrom 65]
- The agent receives noisy observations of the underlying world state
- Need to remember previous observations in order to act optimally
- More difficult, but there are DP algorithms [Smallwood & Sondik 73]

Decentralized POMDP



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

Relationship with other models



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

Modeling assumptions

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

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.*

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 provides a noisy observation of which door the tiger is behind.

Multiagent tiger problem as a Dec-POMDP

$I = \{1, 2\}$

$S = \{SL, SR\}$

$A_1 = A_2 = \{\text{OpenLeft, OpenRight, Listen}\}$

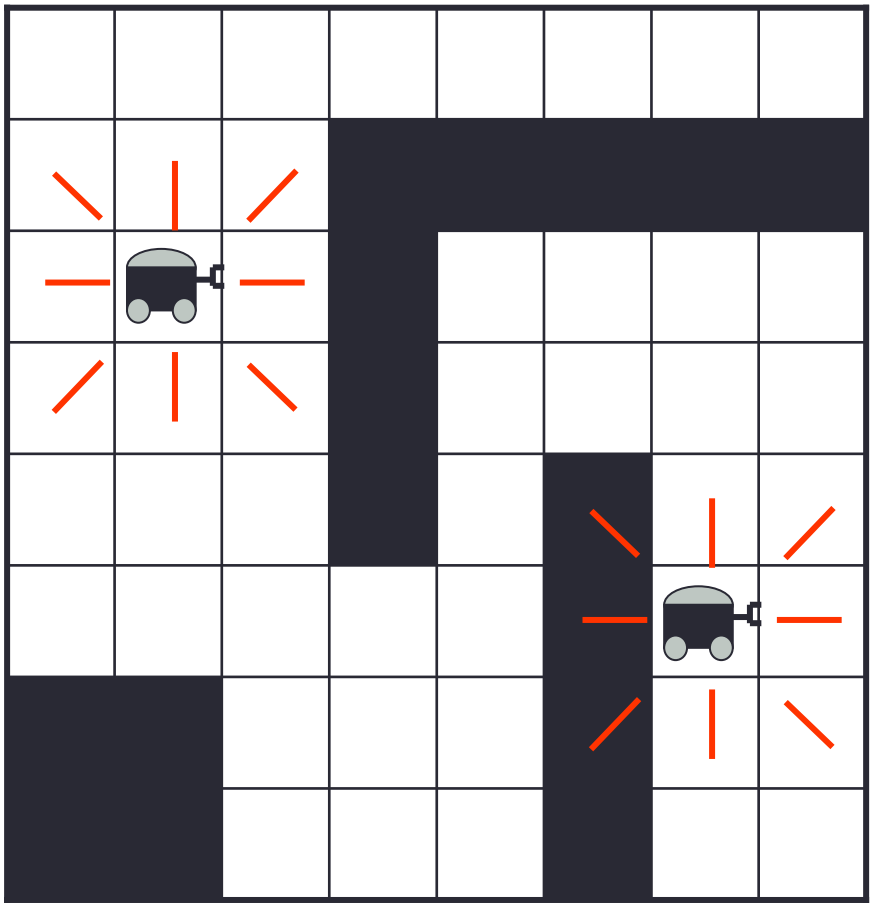
Action/Transition	SL → SL	SL → SR	SR → SR	SR → SL
<OpenRight,*>	0.5	0.5	0.5	0.5
<OpenLeft,*>	0.5	0.5	0.5	0.5
<*,OpenLeft>	0.5	0.5	0.5	0.5
<*,OpenRight>	0.5	0.5	0.5	0.5
<Listen,Listen>	1.0	0.0	1.0	0.0

$\Omega_1 = \Omega_2 = \{HL, HR\}$

Action	State	HL	HR
<Listen,Listen>	SL	0.85	0.15
<Listen,Listen>	SR	0.15	0.85
<OpenRight,*>	*	0.5	0.5
<OpenLeft,*>	*	0.5	0.5
<*,OpenLeft>	*	0.5	0.5
<*,OpenRight>	*	0.5	0.5

Action/State	SL	SR
<OpenRight,OpenRight>	+20	-50
<OpenLeft,OpenLeft>	-50	+20
<OpenRight,OpenLeft>	-100	-100
<OpenLeft,OpenRight>	-100	-100
<Listen,Listen>	-2	-2
<Listen,OpenRight>	+9	-101
<OpenRight,Listen>	+9	-101
<Listen,OpenLeft>	-101	+9
<OpenLeft,Listen>	-101	+9

Example: 2-Agent Navigation



States: grid cell pairs

Actions: move ⇧, ⇩, ⇨, ⇦, stay

Transitions: noisy

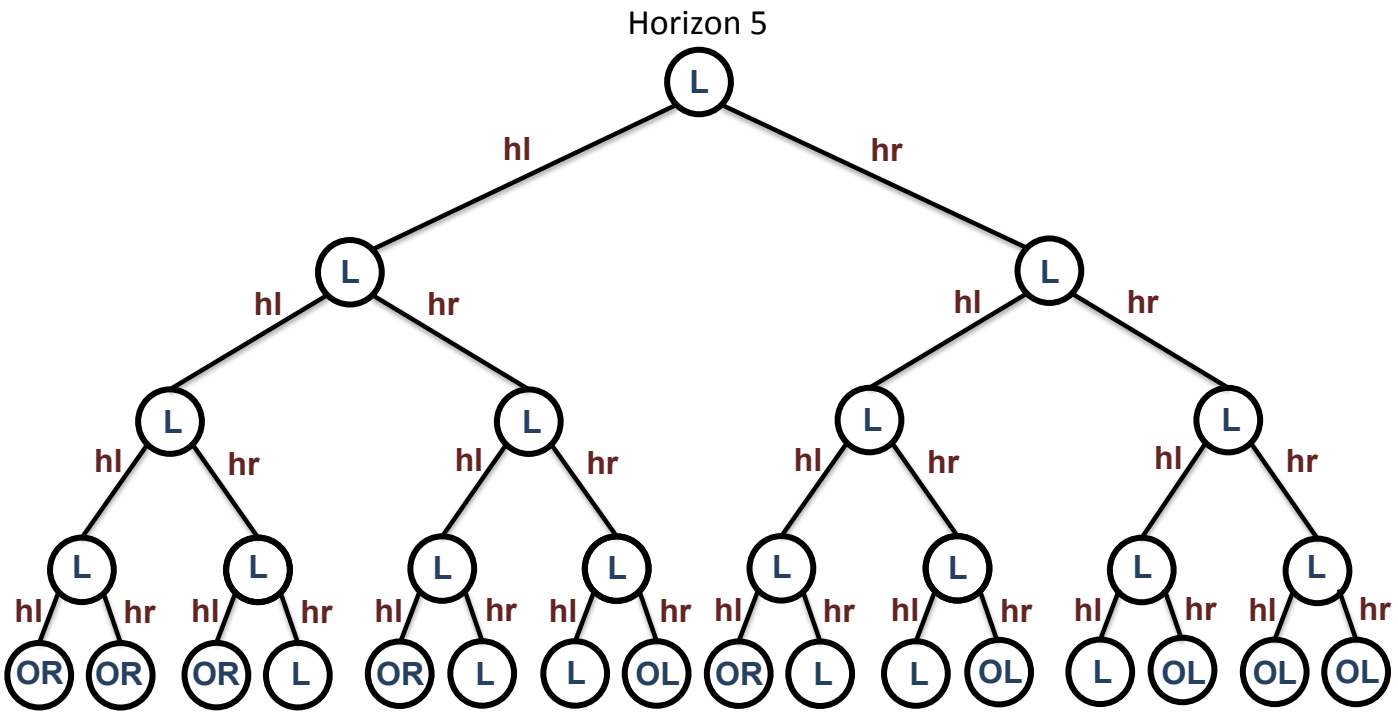
Observations: red lines

Rewards: negative unless sharing the same square

Dec-POMDP solutions

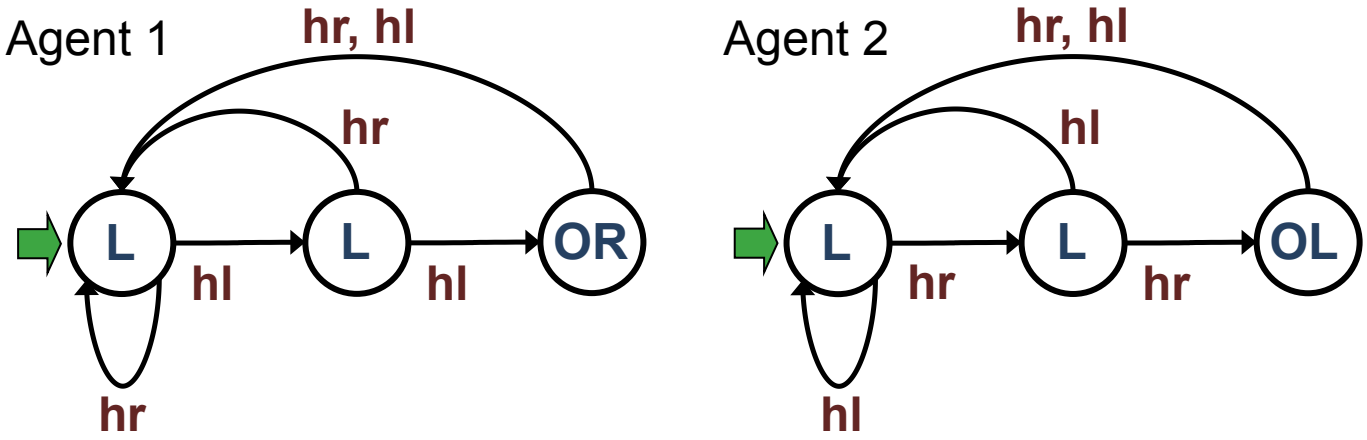
- A local policy for each agent is a mapping from its observation sequences to actions, $\Omega^*_i \rightarrow A_i$
  - State is unknown, so beneficial to remember history
- A joint policy is a local policy for each agent
- Goal is to maximize expected cumulative reward over a finite or infinite horizon
  - For infinite-horizon cannot remember the full observation history
  - In infinite case, a discount factor, γ, is used

Solutions as Policy Trees



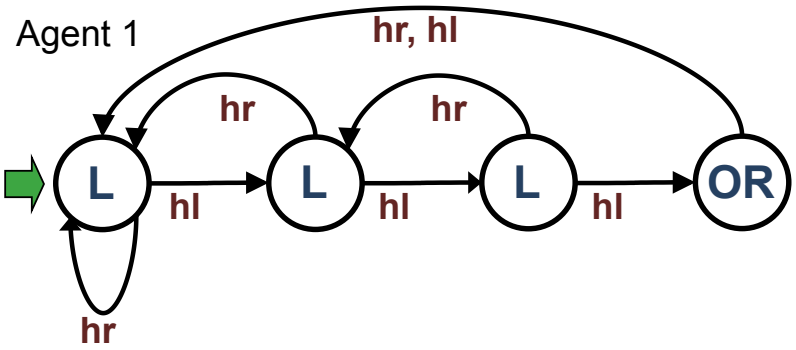
- Each node is labeled with an action and each edge with an observation that could be received
- Policy tree shown above is optimal for the multiagent tiger problem with horizon 5. (Same tree assigned to both agents)

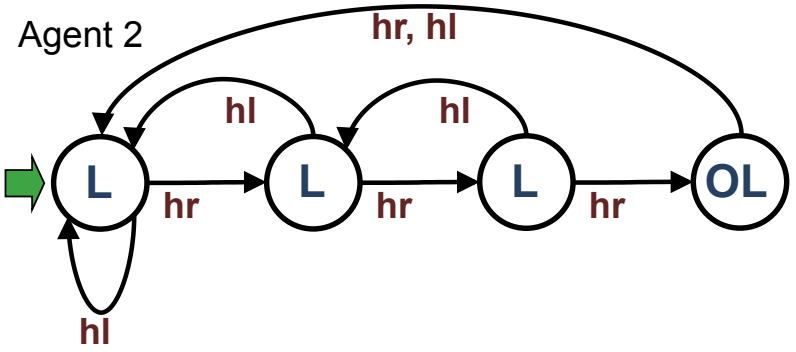
Solutions as Finite-State Controllers



- 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.

Evaluating Solutions

- For a finite-horizon problem with initial state s_0 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_0 and discount factor γ in [0;1), the value of a joint policy δ is

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

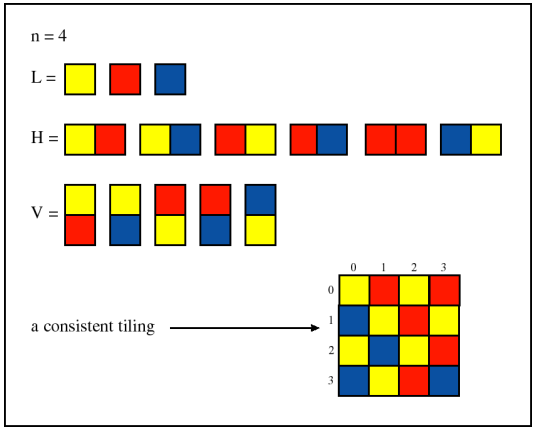
Challenges in solving Dec-POMDPs

- Partial observability makes the problem difficult to solve
- No common state estimate (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 centralized (POMDP) algorithms

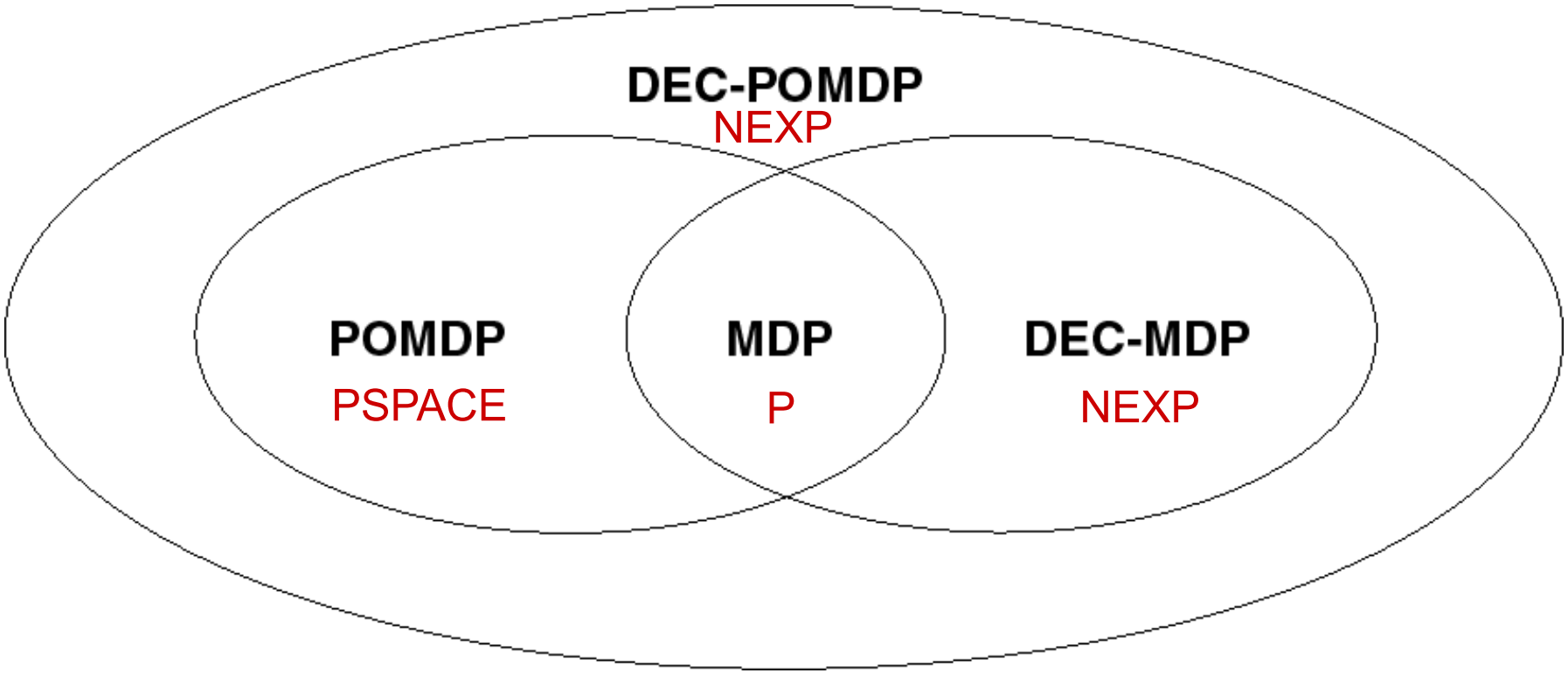
How Hard are DEC-POMDPs?

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?



General complexity results



subclasses and finite horizon complexity results

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 $\vec{o}$.

Optimal methods

- Want to produce the optimal solution for a problem
- That is, optimal horizon *h* tree or ε-optimal controller
- Do this in a bottom up or a top down manner

Bottom up methods

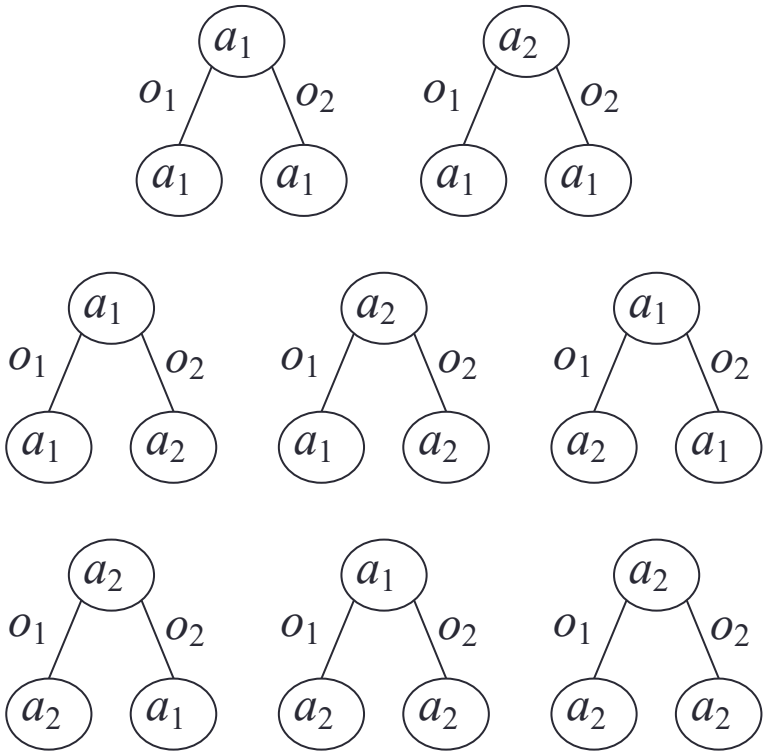
- Build the policies up for each agent simultaneously
- Begin on the last step (single action) and continue until the first
- When done, choose highest value set of trees for any initial state

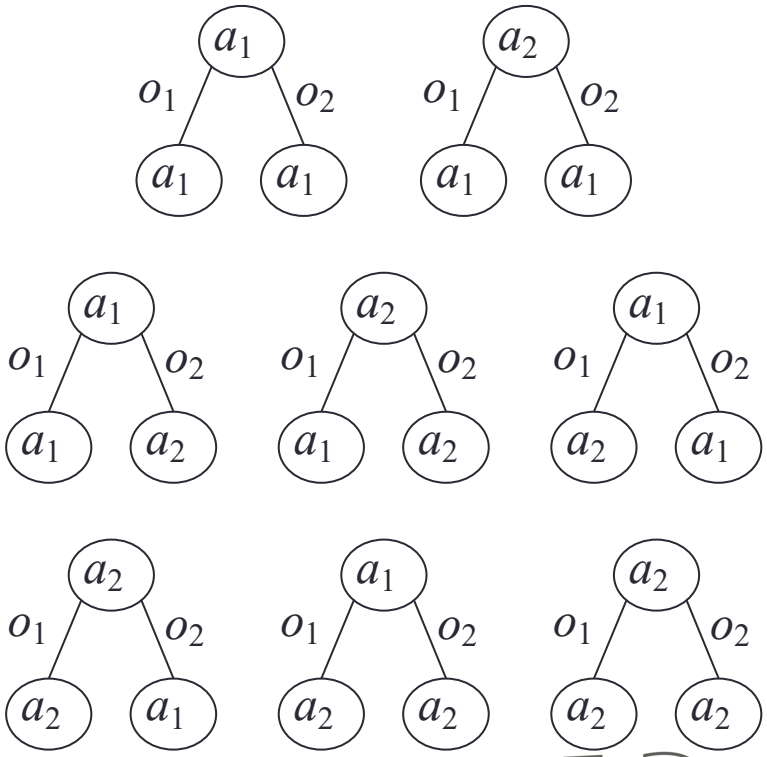
Exhaustive search

- Construct all possible policies for the set of agents
- Do this in a bottom up fashion
- Stop when the desired horizon is reached
- Trivially includes an optimal set of trees when finished
- Choose the set of policies with the highest value at the initial belief

Exhaustive search example (2 agents, 2 actions, 2 observations)

Exhaustive search continued

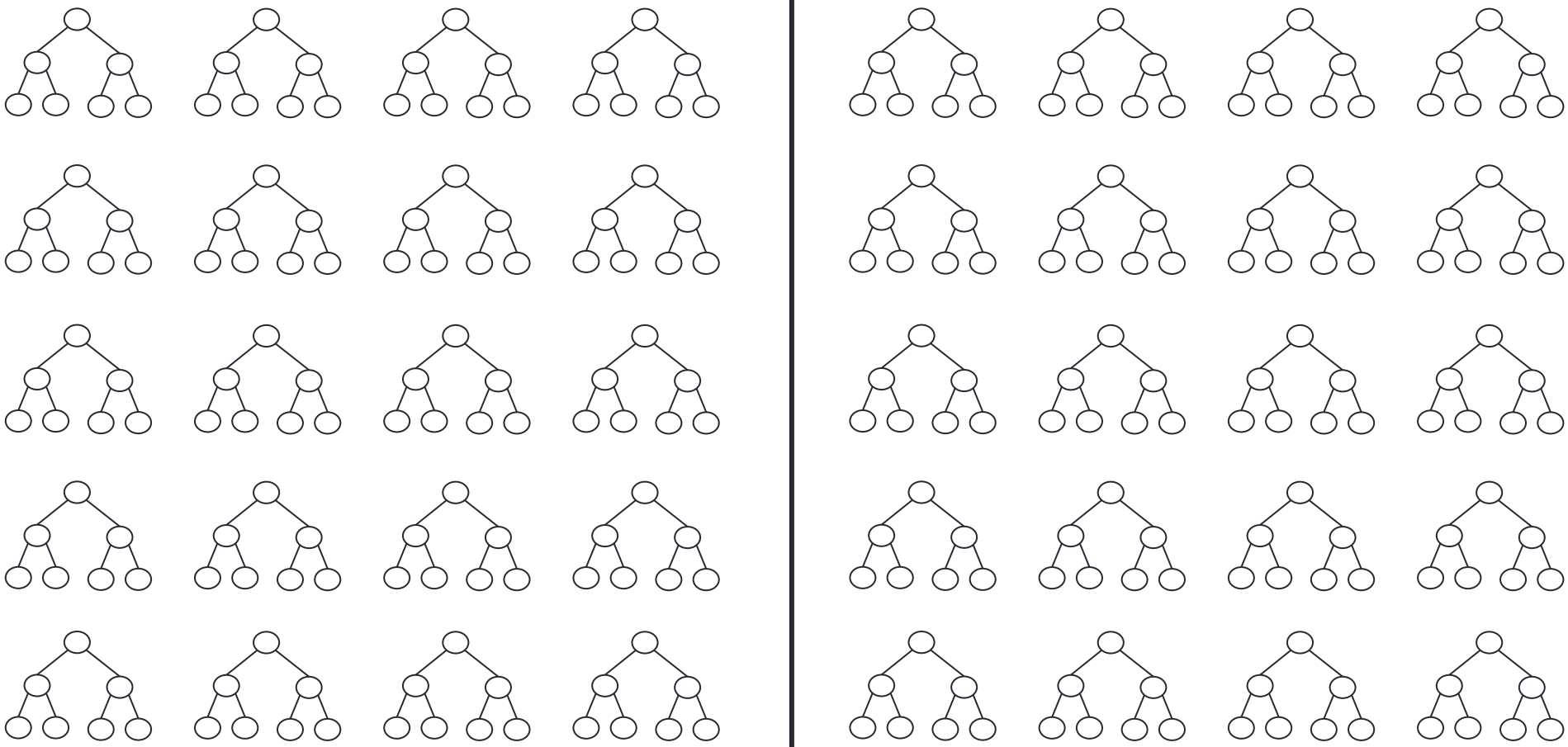




Massachusetts Institute of Technology

CSAIL

Exhaustive search continued



Massachusetts Institute of Technology

CSAIL

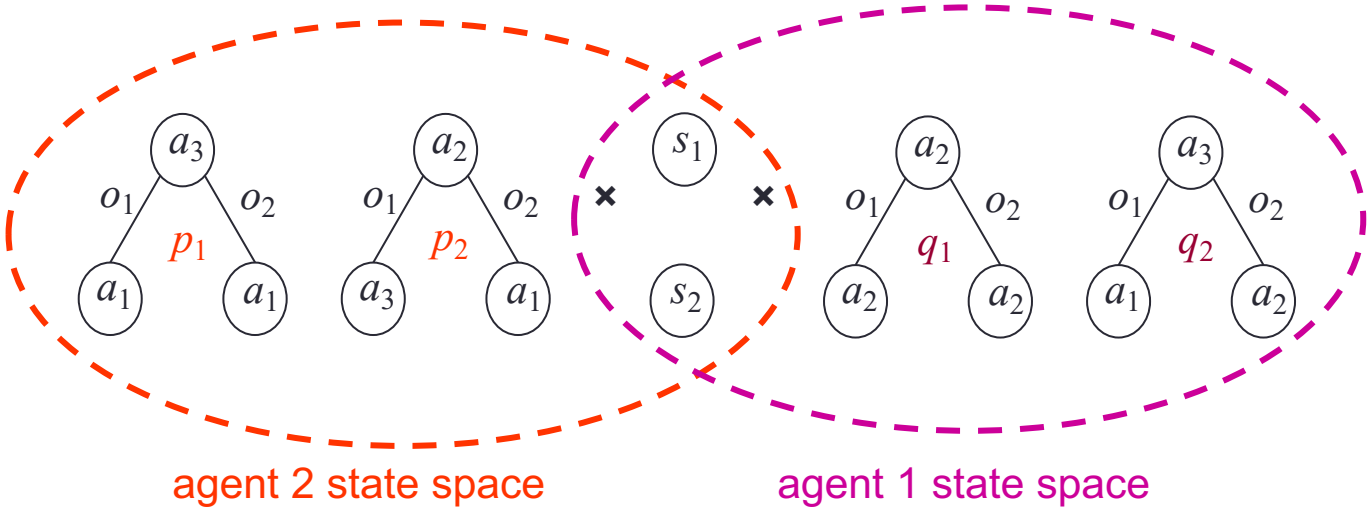
Exhaustive search summary

- Can find an optimal set of trees
- Number of each agent's trees grows exponentially at each step
- Many trees will not contribute to an optimal solution
- Can we reduce the number of trees we consider?

Finite horizon dynamic programming (DP)

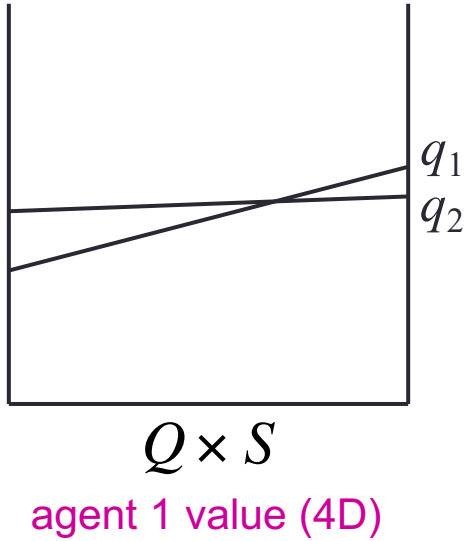
(Hansen et al. 2004)

- Build policy tree sets simultaneously
- Returns optimal policy for any initial state
- Prune dominated policies using a linear program
- This becomes iterative elimination of dominated strategies in POSGs
- For two agents:



agent 2 state space

agent 1 state space



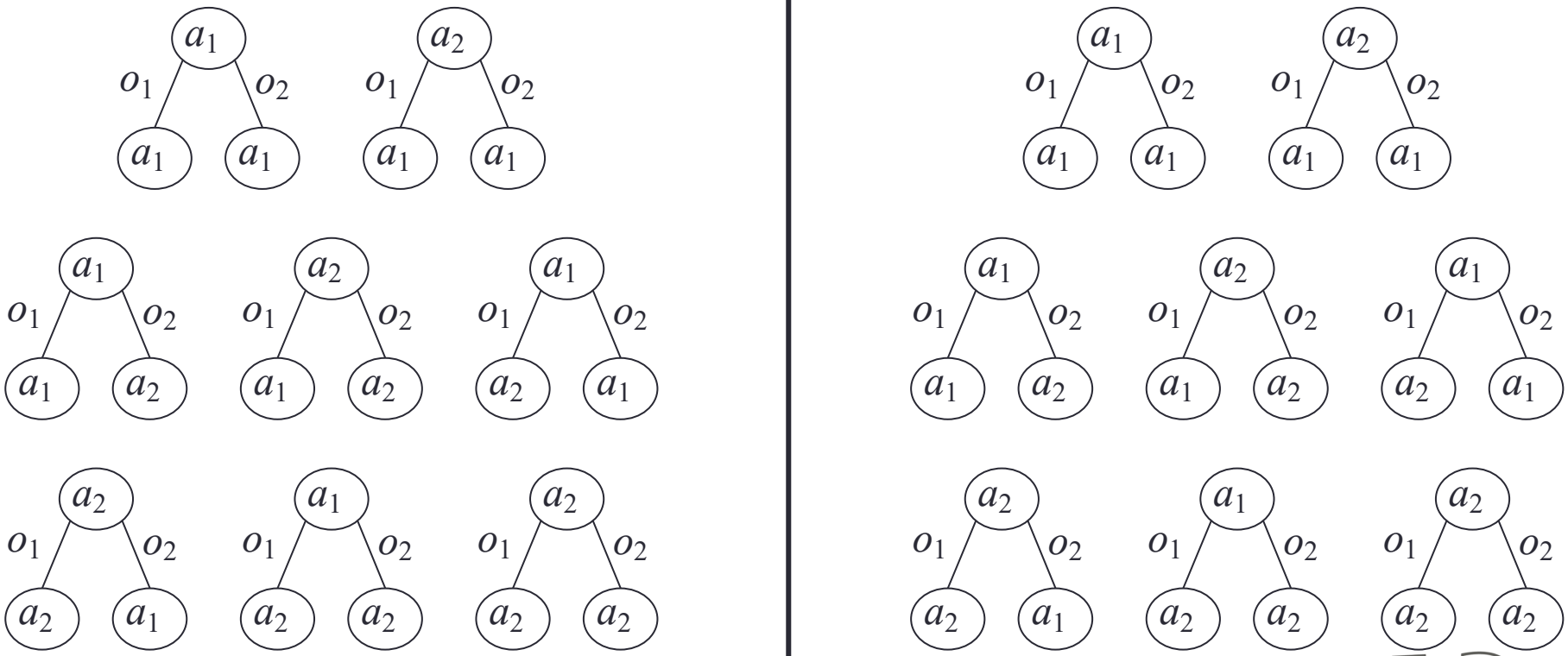
agent 1 value (4D)

Massachusetts Institute of Technology

CSAIL

DP for DEC-POMDPs example (2 agents)

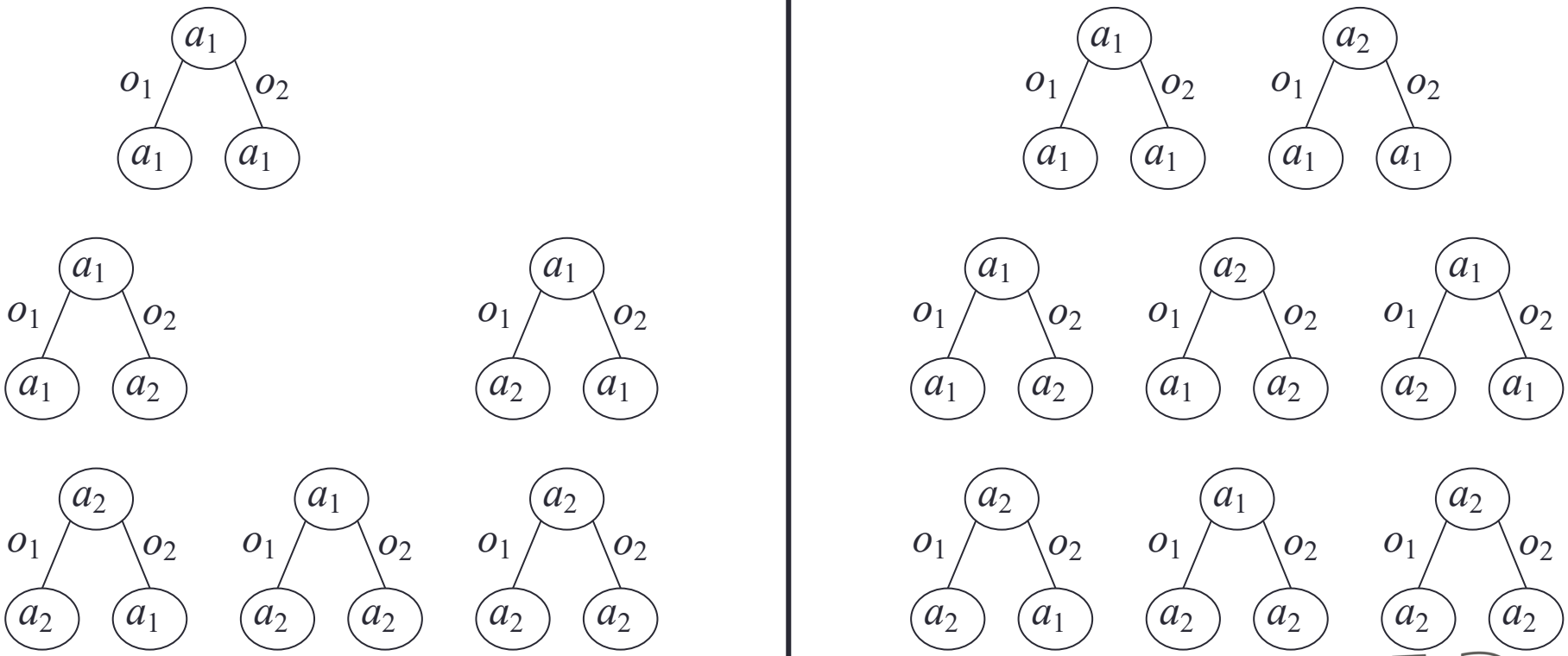
DP for DEC-POMDPs continued



Massachusetts Institute of Technology

CSAIL

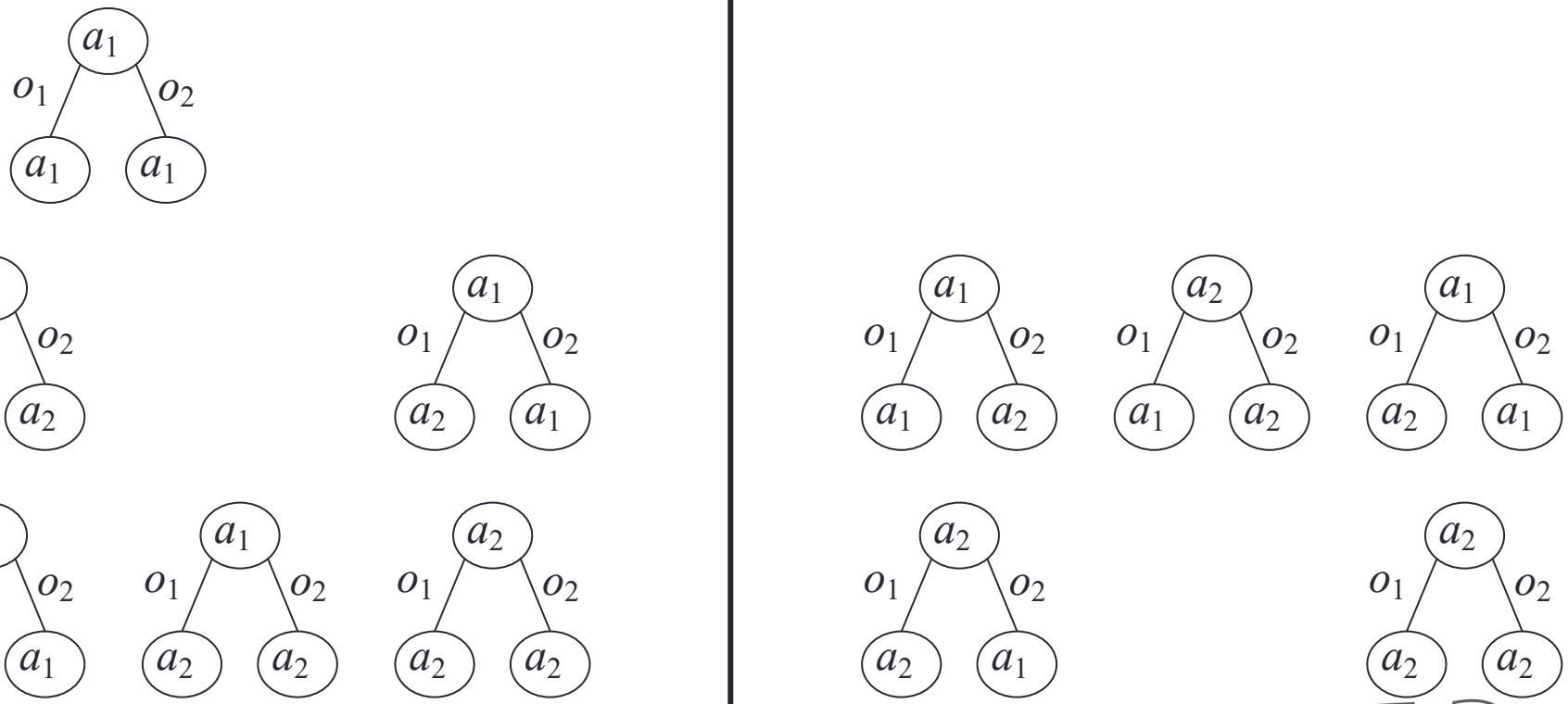
DP for DEC-POMDPs continued



Massachusetts Institute of Technology

CSAIL

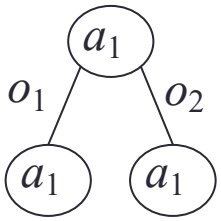
DP for DEC-POMDPs continued

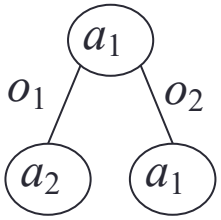


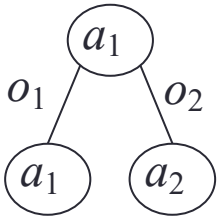
Massachusetts Institute of Technology

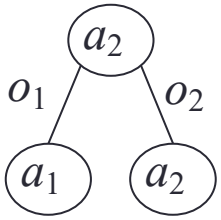
CSAIL

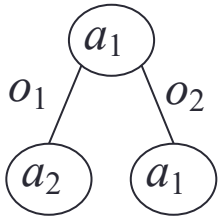
DP for DEC-POMDPs continued

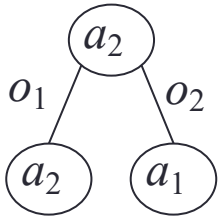


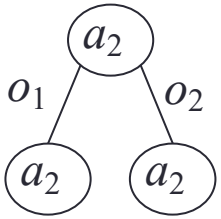


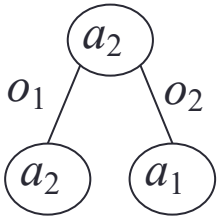


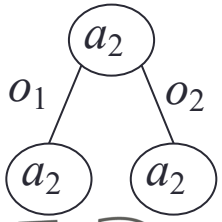








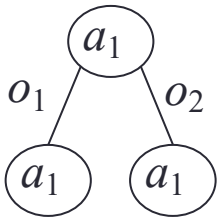


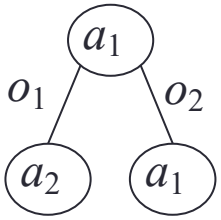


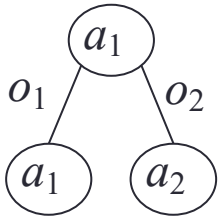
Massachusetts Institute of Technology

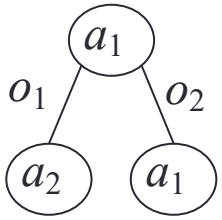
CSAIL

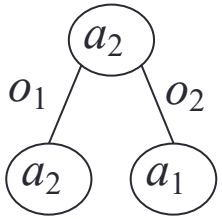
DP for DEC-POMDPs continued

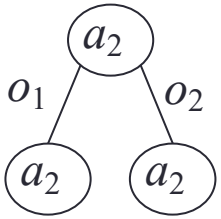


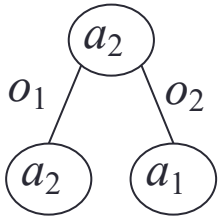


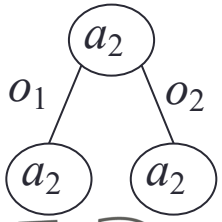








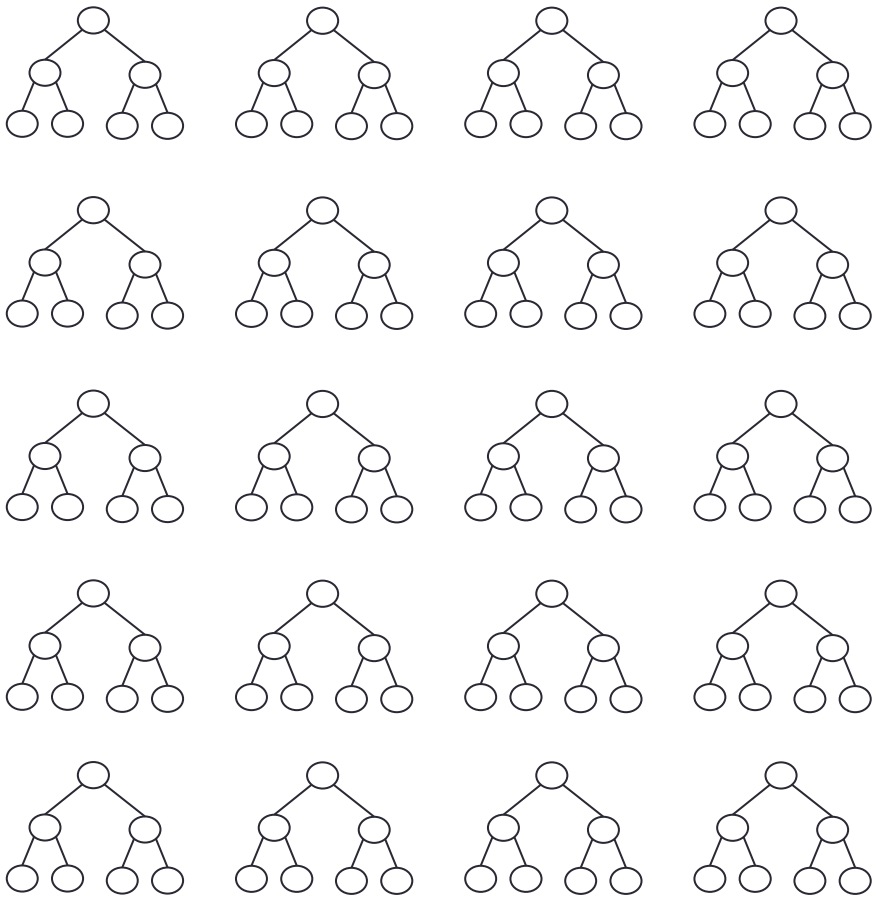


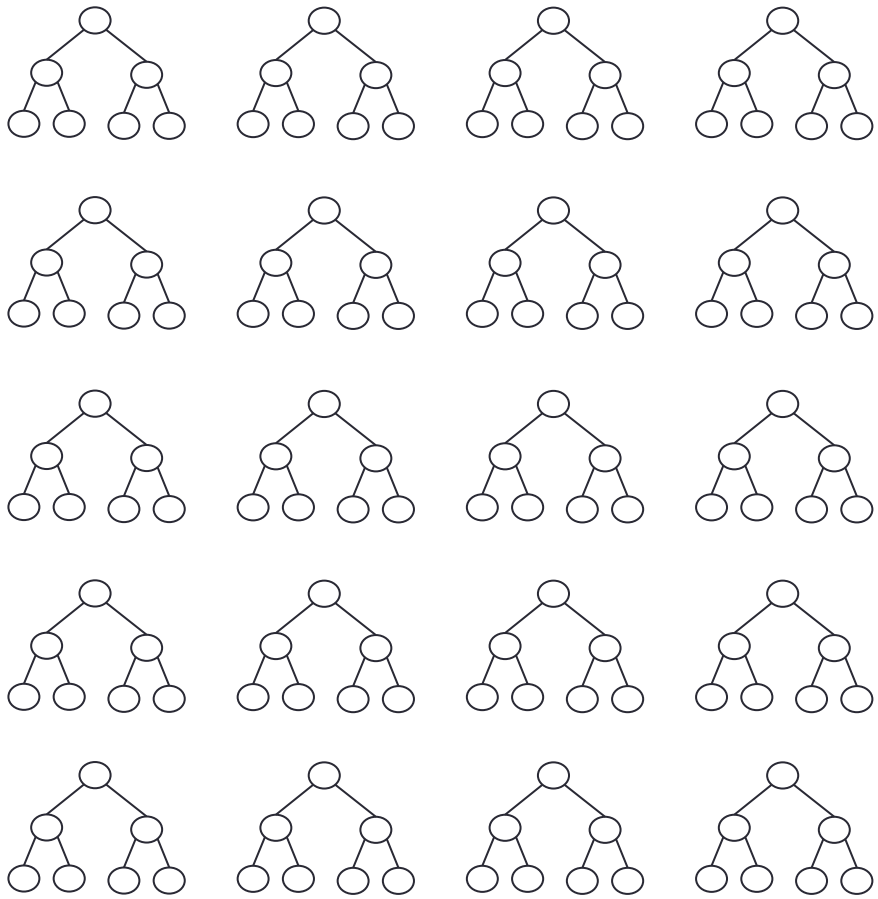


Massachusetts Institute of Technology

CSAIL

DP for DEC-POMDPs continued





Massachusetts Institute of Technology

CSAIL

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

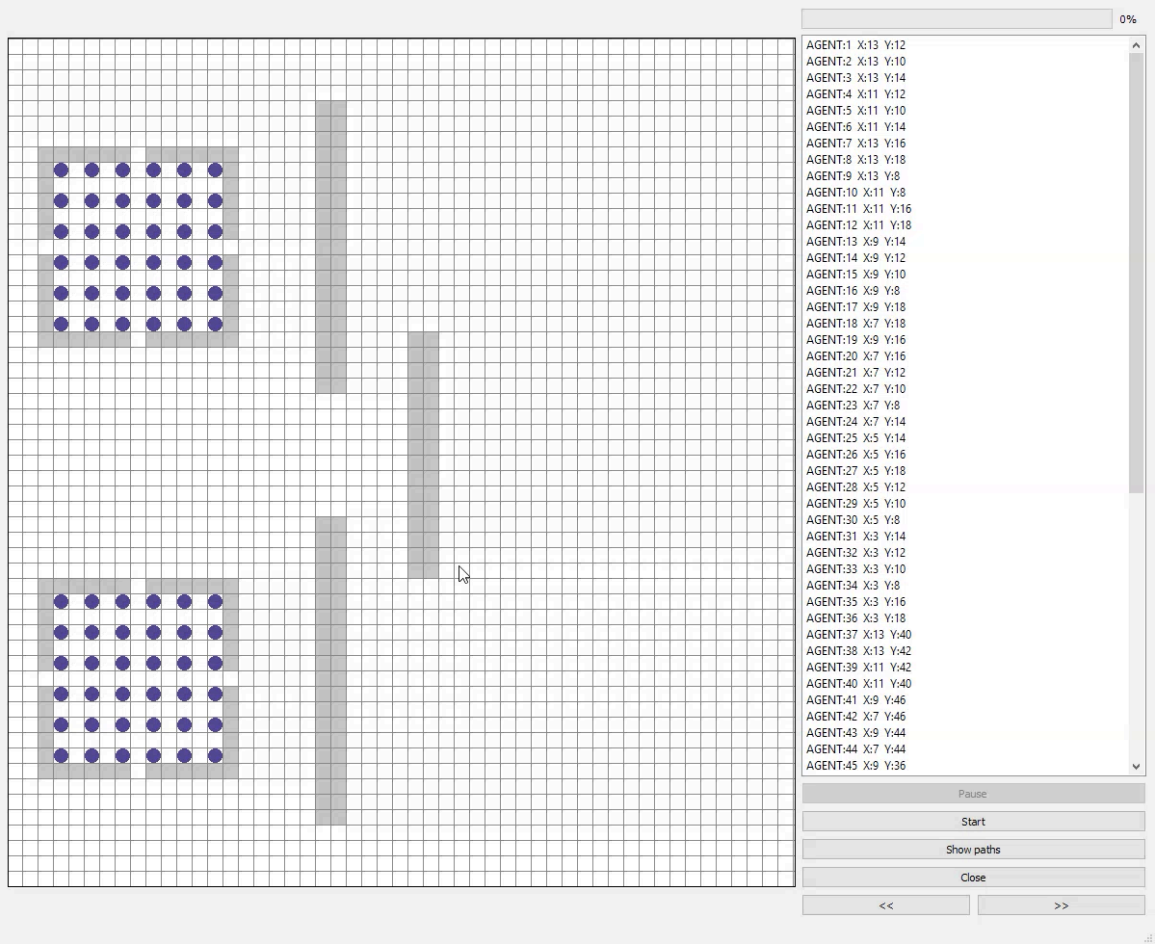
Multiagent planning in spatial environments

Special case of generic multiagent planning

Multiagent path finding

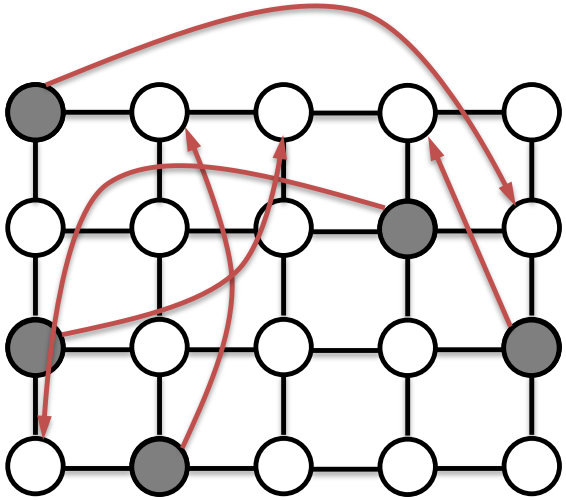
Plans = sequences of movements in an environment (physical or virtual)

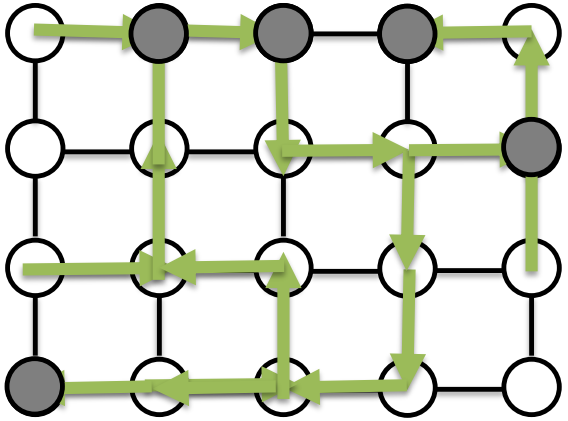
Relevant for applications



https://github.com/PathPlanning/AA-SIPP-m

What is **multi-agent path finding** (MAPF)?





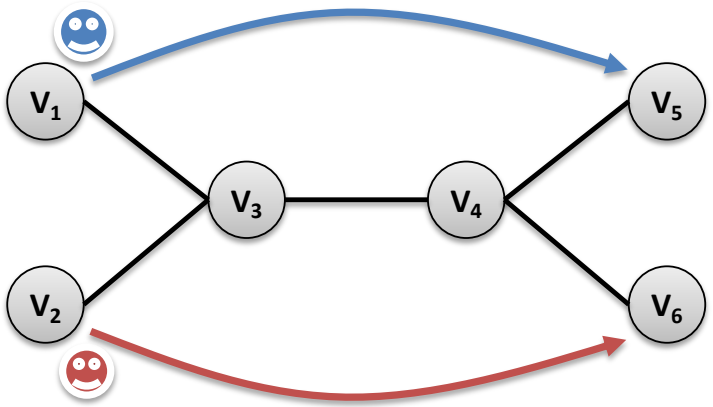
MAPF problem:

Find a **collision-free** plan (path) for each agent

Alternative names:

cooperative path finding (CPF), multi-robot path planning, pebble motion

- a **graph** (directed or undirected)
- a set of **agents**, each agent is assigned to two locations (nodes) in the graph (start, destination)



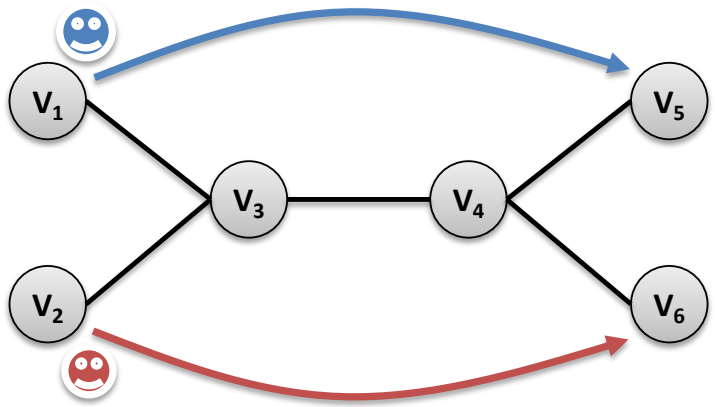
Each agent can perform either **move** (to a neighboring node) or **wait** (in the same node) actions.

- *Typical assumption*:
  - all move and wait actions have identical durations (plans for agents are synchronized)

Plan is a sequence of actions for the agent leading from its start location to its destination.

- The **length of a plan** (for an agent) is defined by the time when the agent reaches its destination and does not leave it anymore.

Find **plans** for all agents such that the plans **do not collide in time and space** (no two agents are at the same location at the same time).

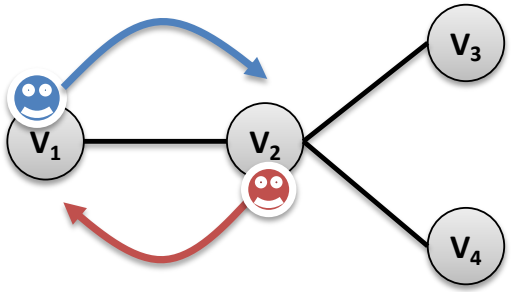


time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	wait $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_3}$	move $\mathbf{v_4}$
3	move $\mathbf{v_4}$	move $\mathbf{v_6}$
4	move $\mathbf{v_5}$	wait $\mathbf{v_6}$

Some necessary **conditions for plan existence**:

- no two agents are at the same start node
- no two agents share the same destination node (unless an agent disappears when reaching its destination)
- the number of agents is strictly smaller than the number of nodes

No-swap constraint



Agent at $\mathbf{v_i}$ cannot perform **move** $\mathbf{v_j}$ at the same time when agent at $\mathbf{v_j}$ performs **move** $\mathbf{v_i}$

Agents may swap position

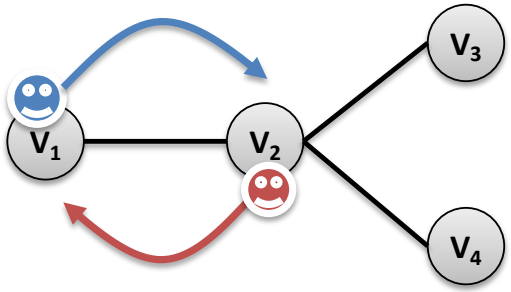
time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents use the same edge at the same time!

Swap is not allowed.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

No-train constraint



Agent at $\mathbf{v_i}$ cannot perform **move** $\mathbf{v_j}$ if there is another agent at $\mathbf{v_j}$

Agent can approach a node that is currently occupied but will be free before arrival.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents form a **train**.

Trains may be forbidden.

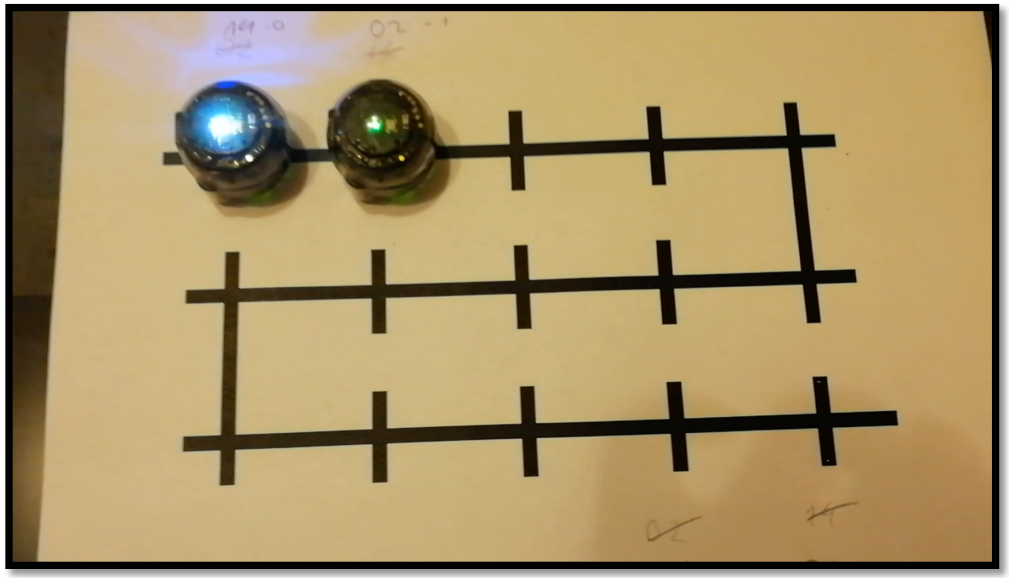
time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	*wait* v_1	move $\mathbf{v_3}$
2	move $\mathbf{v_2}$	*wait* v_3
3	move $\mathbf{v_4}$	*wait* v_3
4	*wait* v_4	move $\mathbf{v_2}$
5	*wait* v_4	move $\mathbf{v_1}$
6	move $\mathbf{v_2}$	*wait* v_1

If any agent is delayed then trains may cause collisions during execution.

To prevent such collisions we may introduce more space between agents.

k-robustness

An agent can visit a node, if that node has not been occupied in recent *k* steps.



1-robustness covers both no-swap and no-train constraints

- No plan (path) has a cycle.
- No two plans (paths) visit the same same location.
- Waiting is not allowed.
- Some specific locations must be visited.
- …

How to measure quality of plans?

Two typical criteria (to minimize):

- **Makespan**
  - distance between the start time of the first agent and the completion time of the last agent
  - maximum of lengths of plans (end times)
- **Sum of costs (SOC)**
  - sum of lengths of plans (end times)

time	agent 1	agent 2
0	v_1	v_2
1	wait v_1	move v_3
2	move v_3	move v_4
3	move v_4	move v_6
4	move v_5	wait v_6

Makespan = 4
SOC = 7

Optimal single agent path finding is tractable.

- e.g. Dijkstra's algorithm

Sub-optimal multi-agent path finding (with two free unoccupied nodes) is tractable.

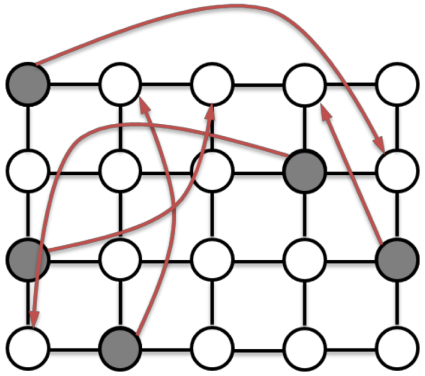
- e.g. algorithm Push and Rotate

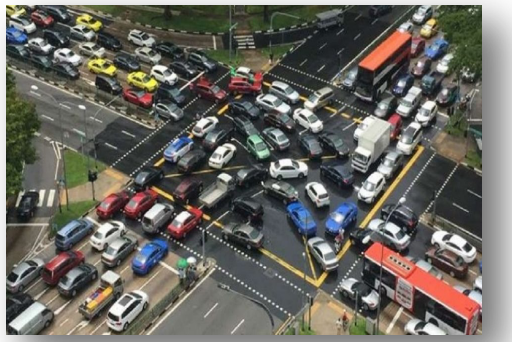
MAPF, where agents have joint goal nodes (it does not matter which agent reaches which goal) is tractable.

- reduction to min-cost flow problem

Optimal (makespan, SOC) multi-agent path finding is **NP-hard**.

Applications





Offline MAPF

Online MAPF

	Warehouse	Intersection
Fixed set of agents	Fixed set of agents	Sequence of agents
One task per agent	Sequence of tasks	One task per agent

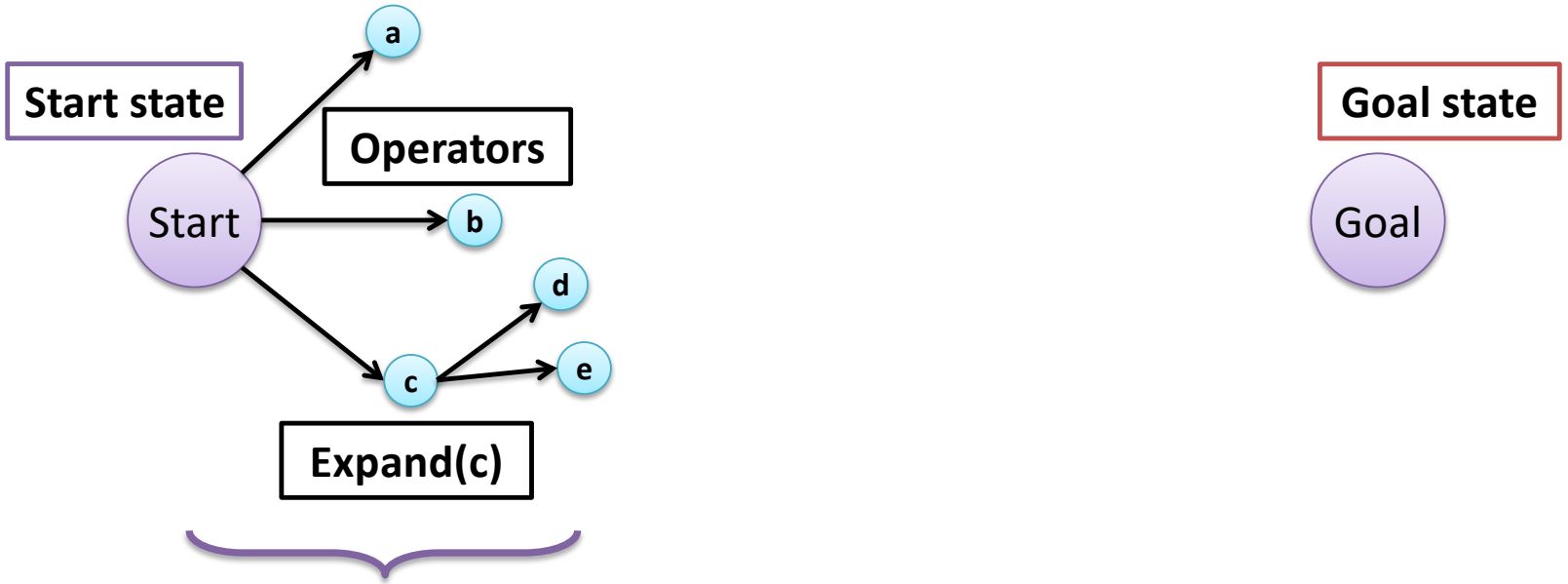
Search-based techniques

- state-space search (A*)
  - state = location of agents at nodes
  - transition = performing one action for each agent
- conflict-based search

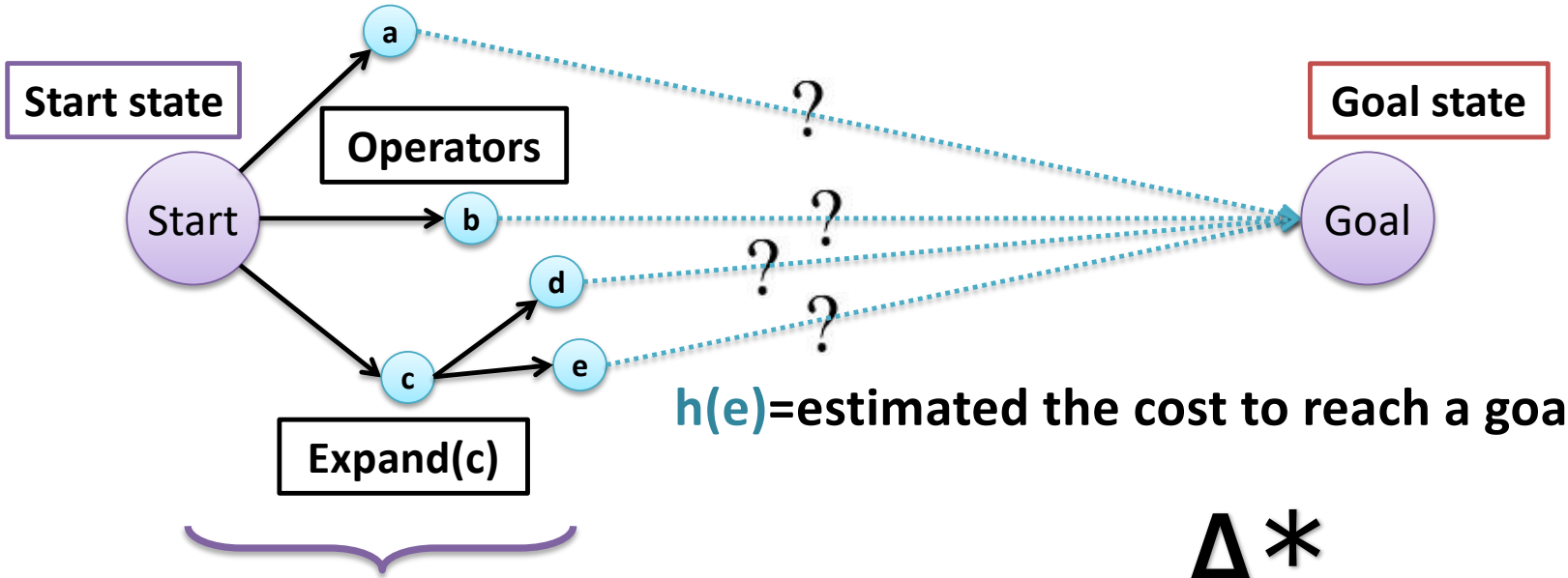
Reduction-based techniques

- translate the problem to another formalism (SAT/CSP/ASP …)

To expand or not expand, this is the question



To expand or not expand, this is the question

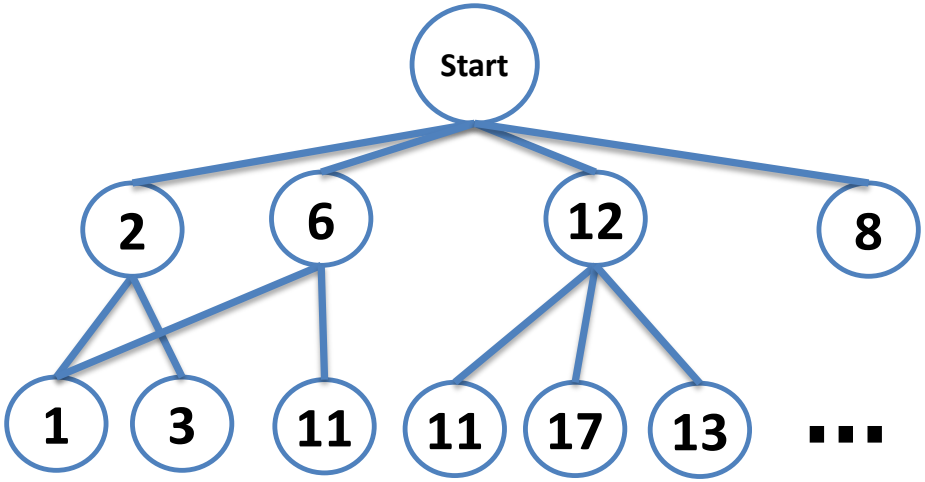


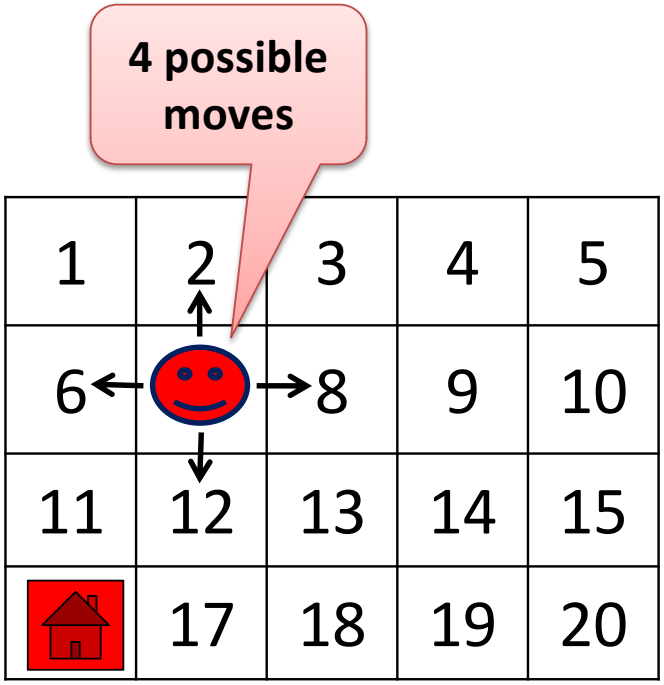
A*

- *Expands min g+h*
- *Returns optimal solutions*
- *"Optimally effective"*

Solving Multi-Agent Path Finding with Search

	Suboptimal	Optimal
Incomplete	?	?
Complete	?	?



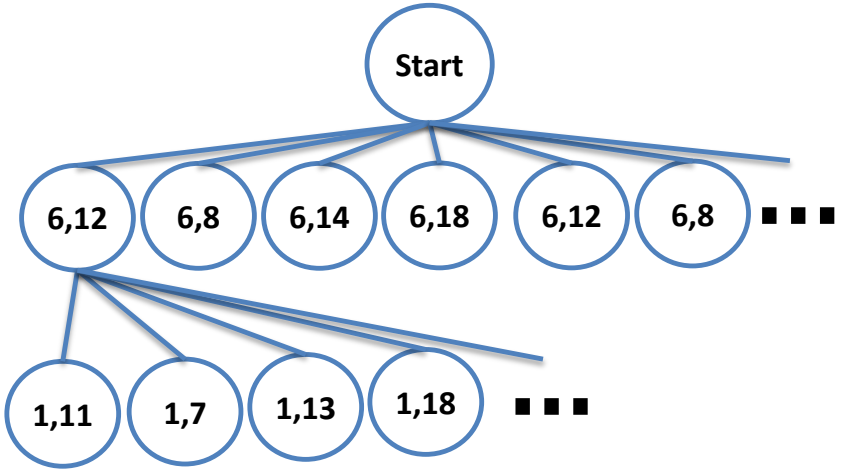


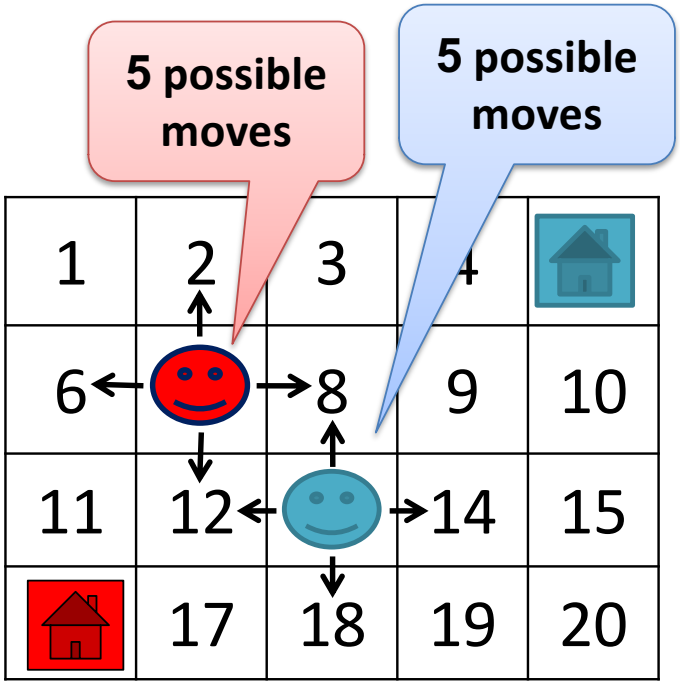
Search problem properties

- Number of states = 20
- Branching factor = 4

Classical search problem!

25 Possible moves! = 5 x 5

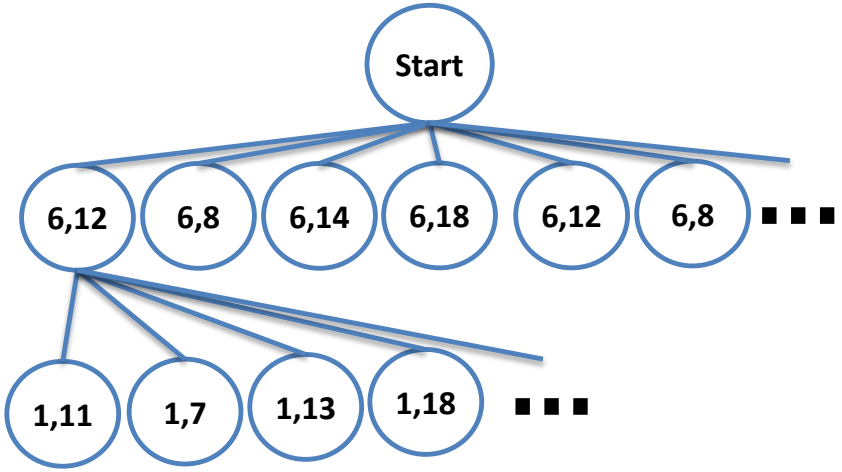


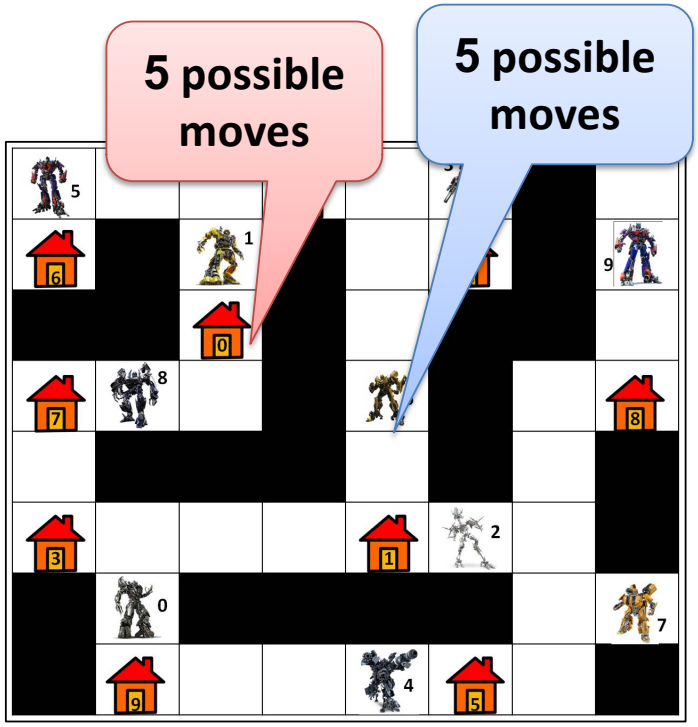


Search problem properties

- Number of states = ?
- Branching factor = ?

25 Possible moves! = 5 x 5





Search problem properties

- Number of states = 20^2
- Branching factor = 5^2

What about k agents?

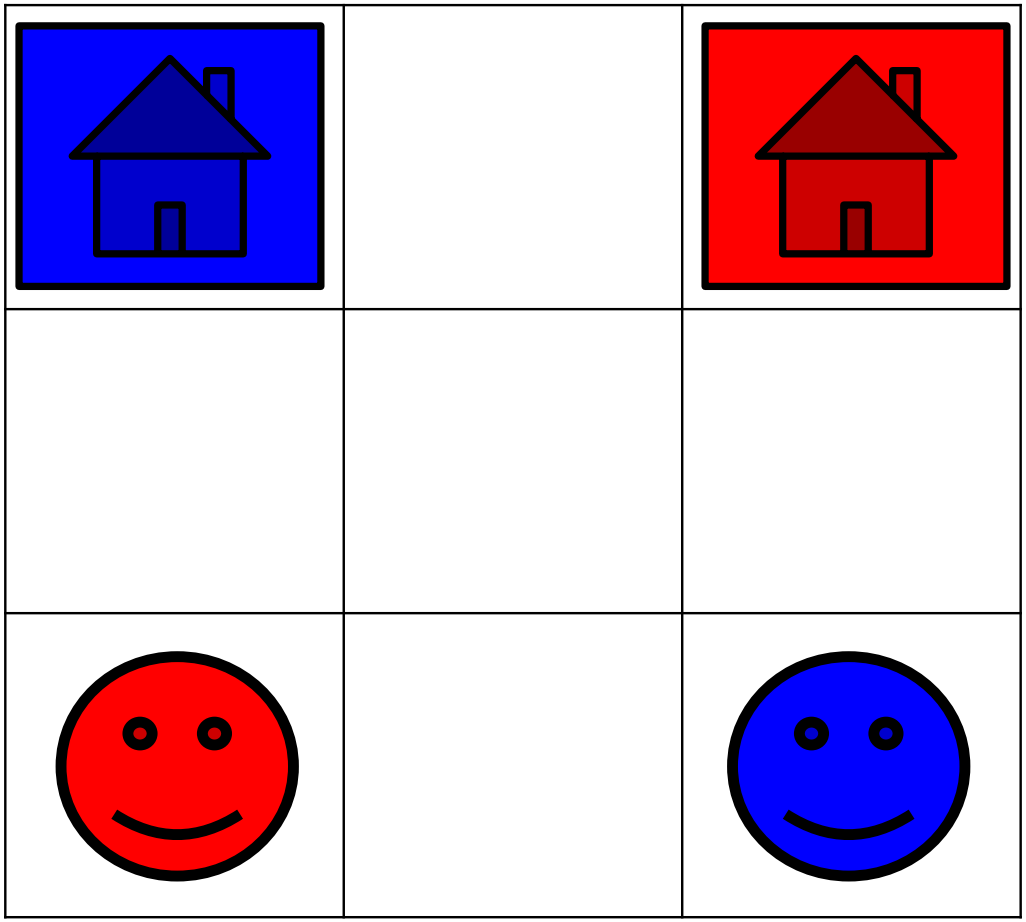
Key idea:

Plan for each agent **separately**

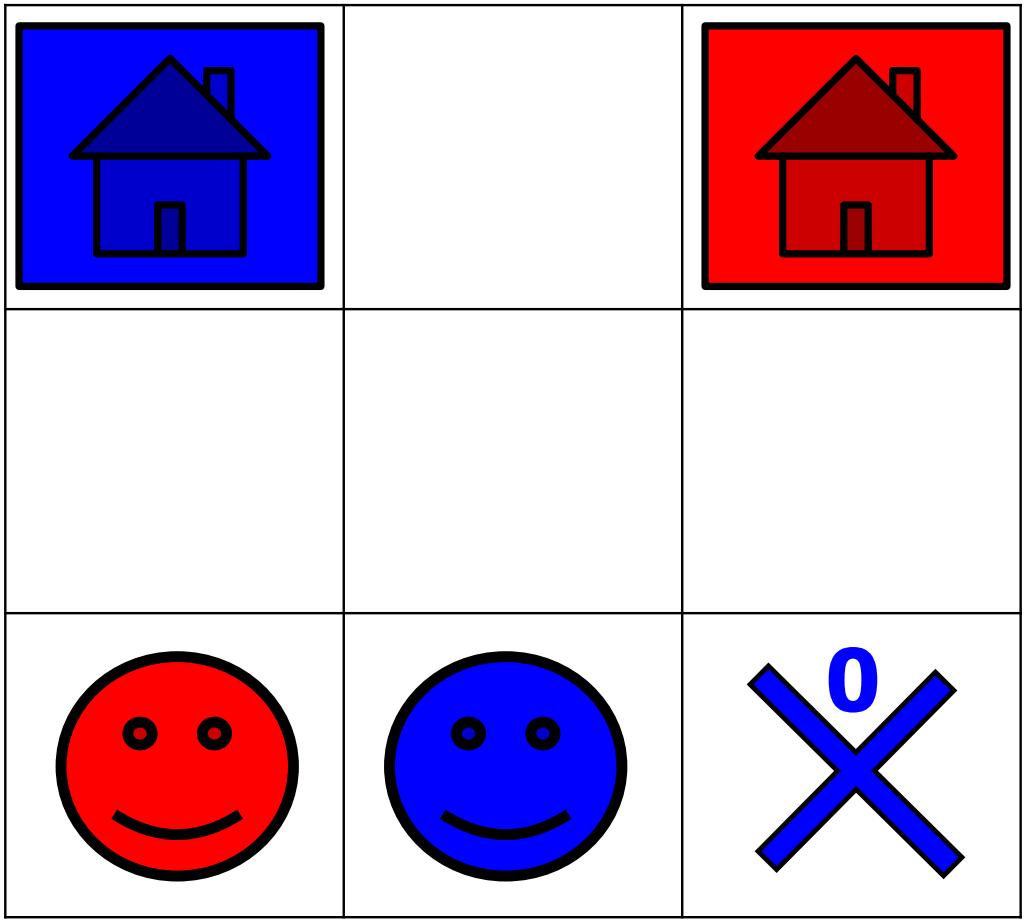
Challenge:

Maintaining **soundness**, **completeness**, and **optimality**

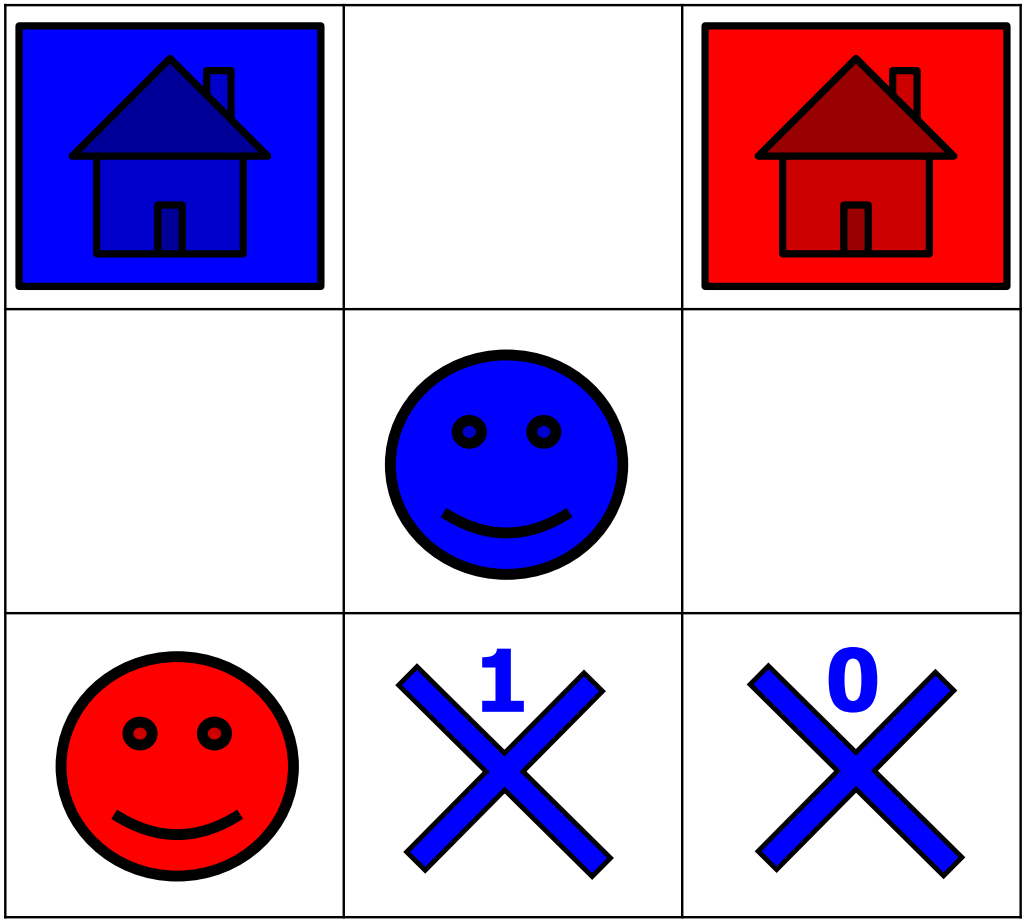
- Step 1: Plan **blue**



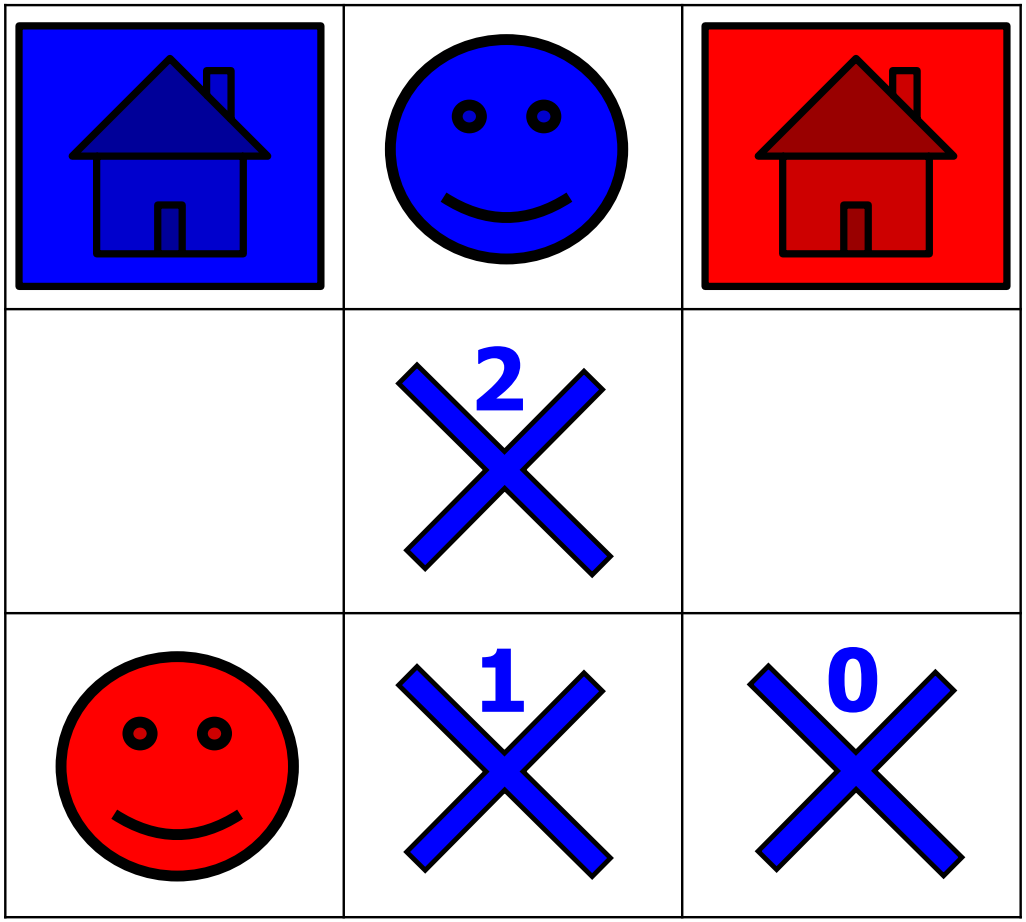
- Step 1: Plan **blue**



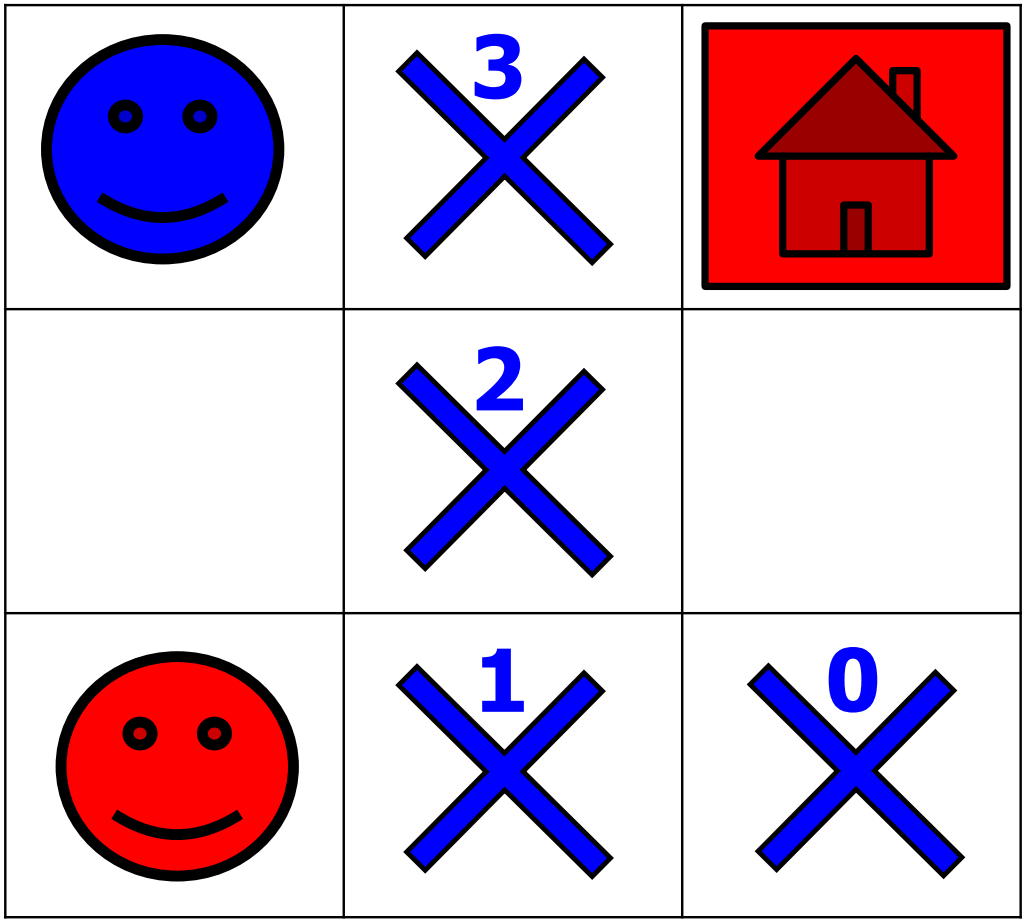
- Step 1: Plan **blue**



- Step 1: Plan **blue**

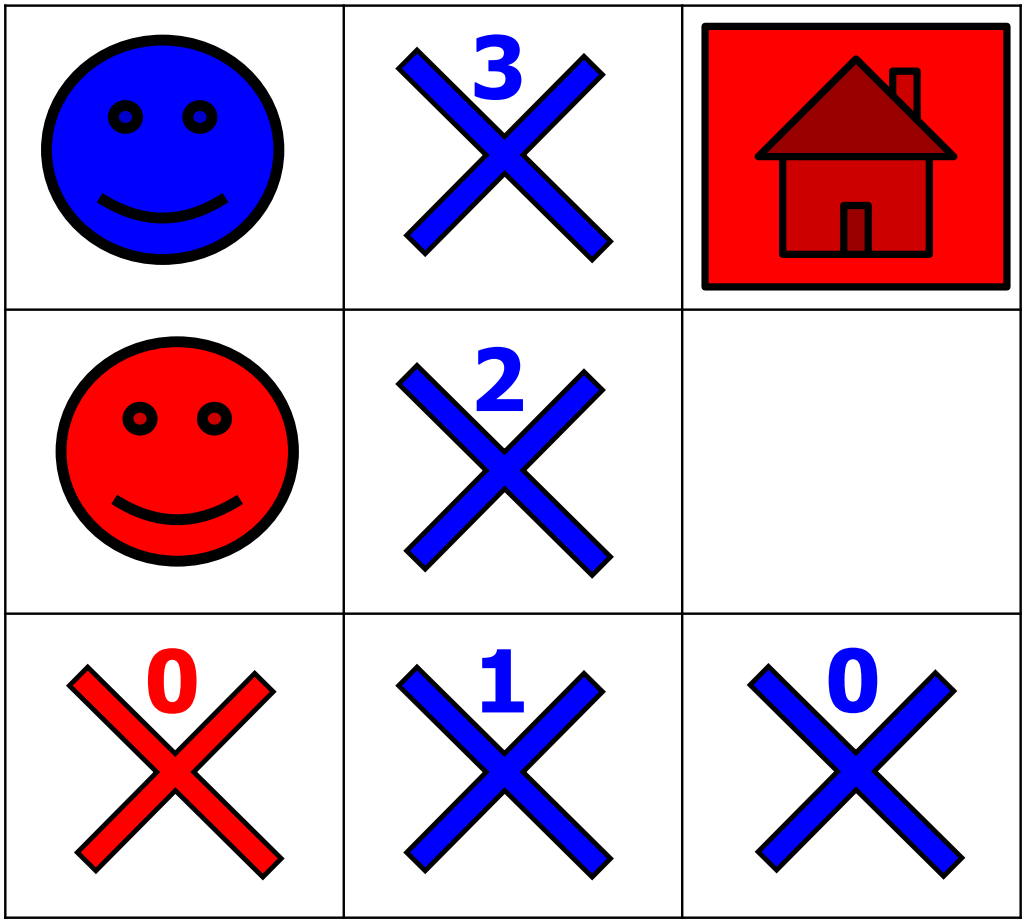


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

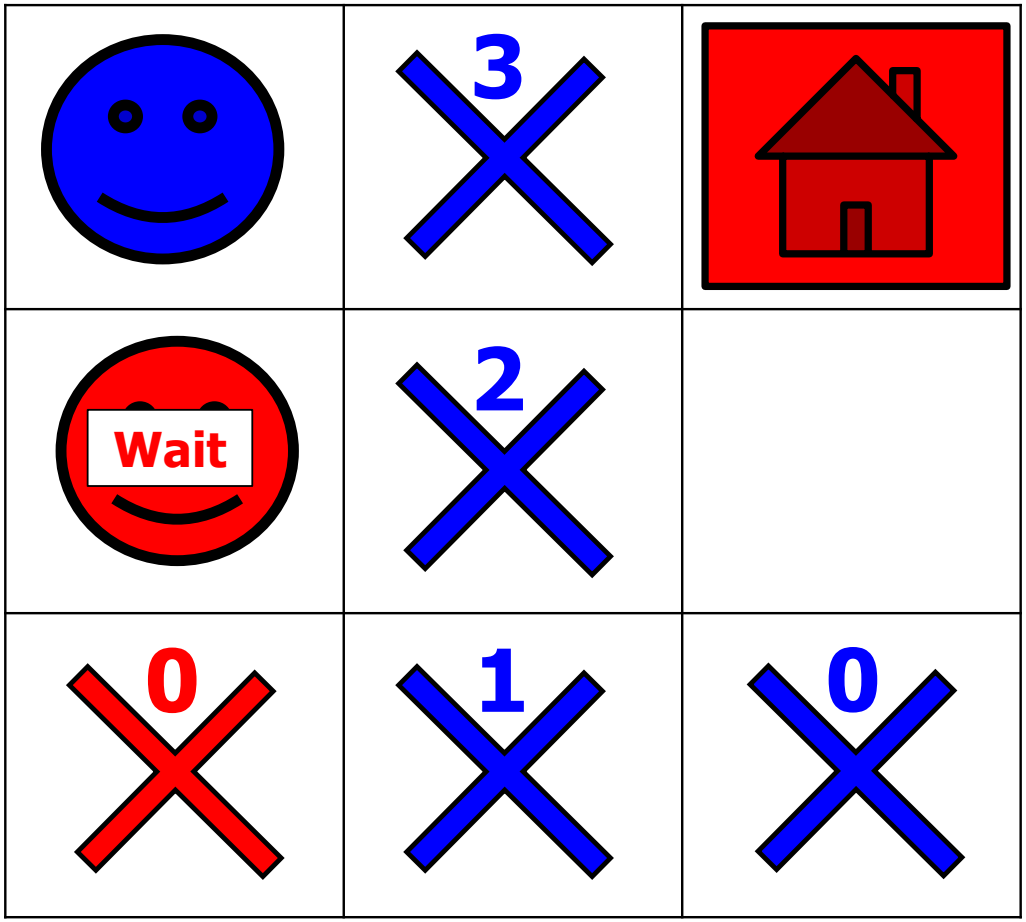


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

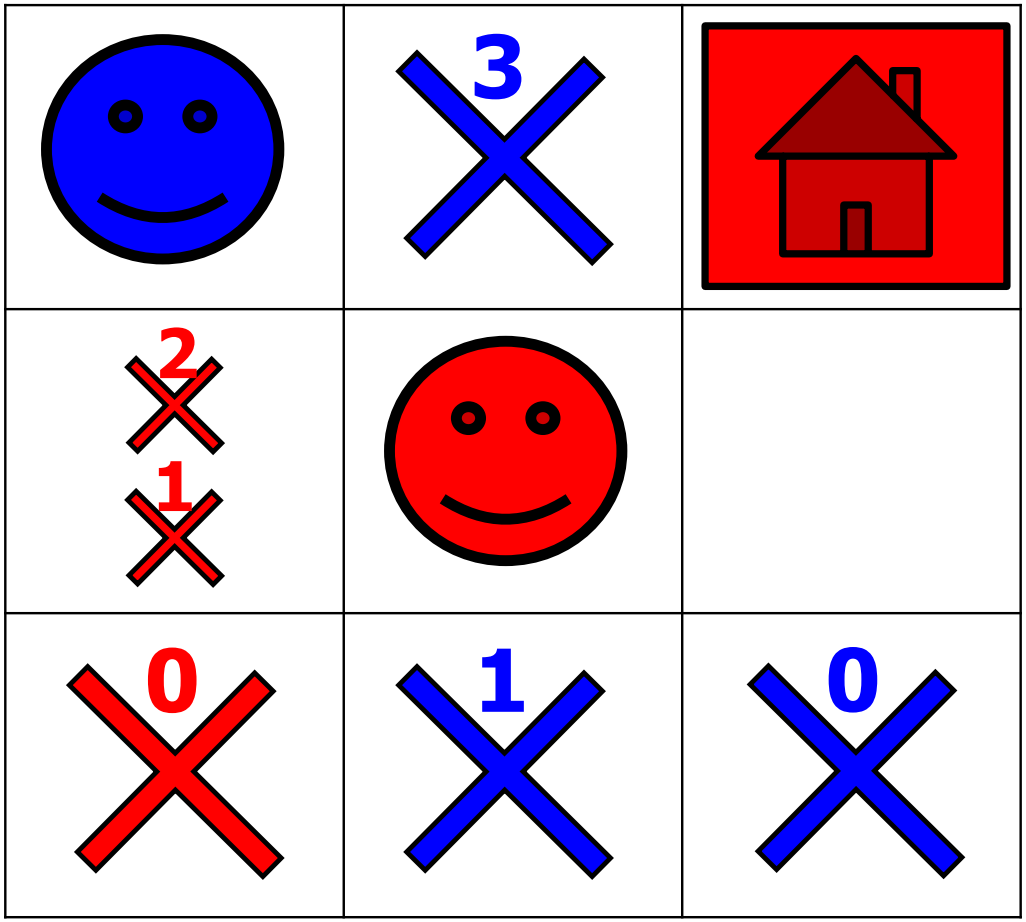
  avoid blue's plan



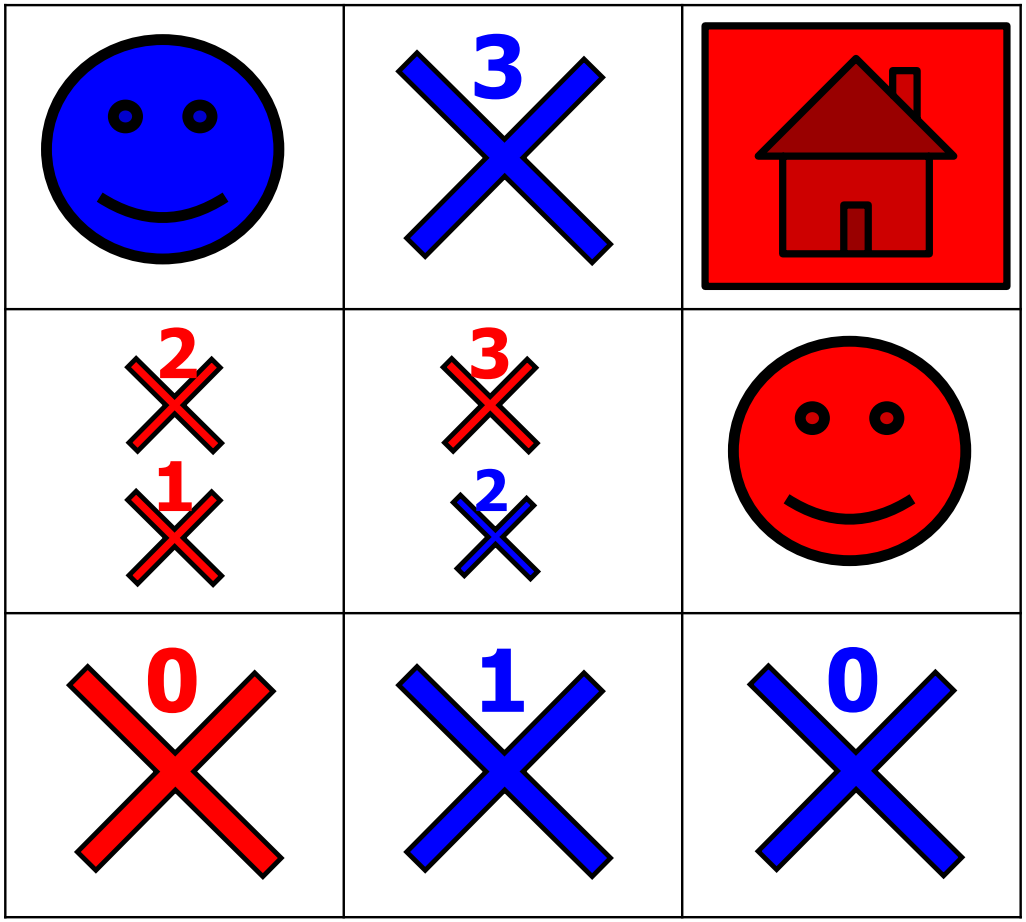
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**



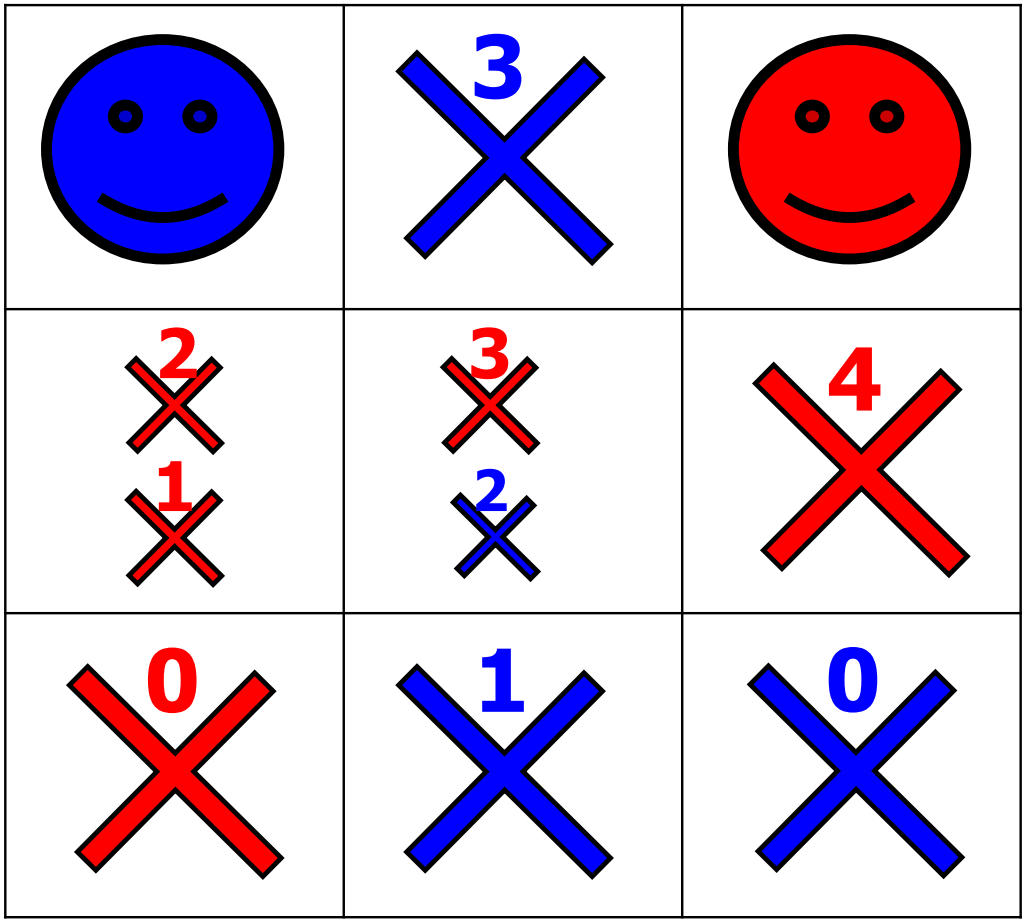
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**



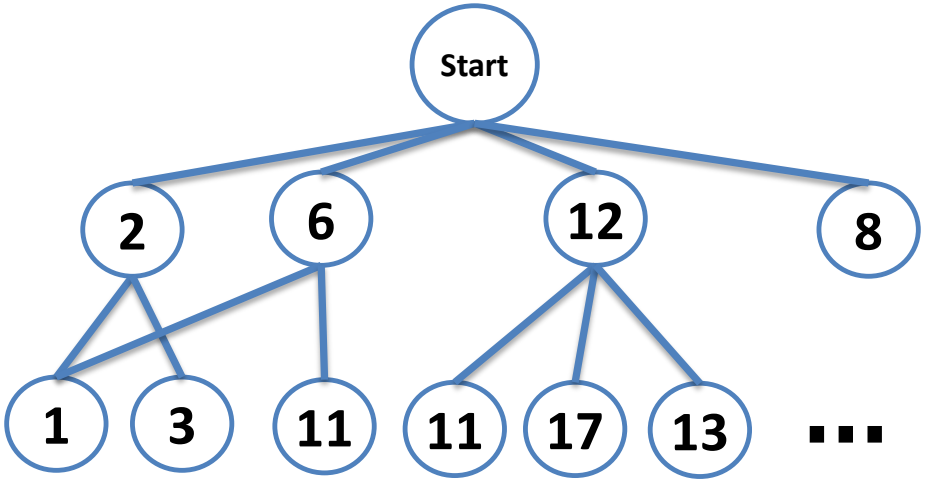
- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**

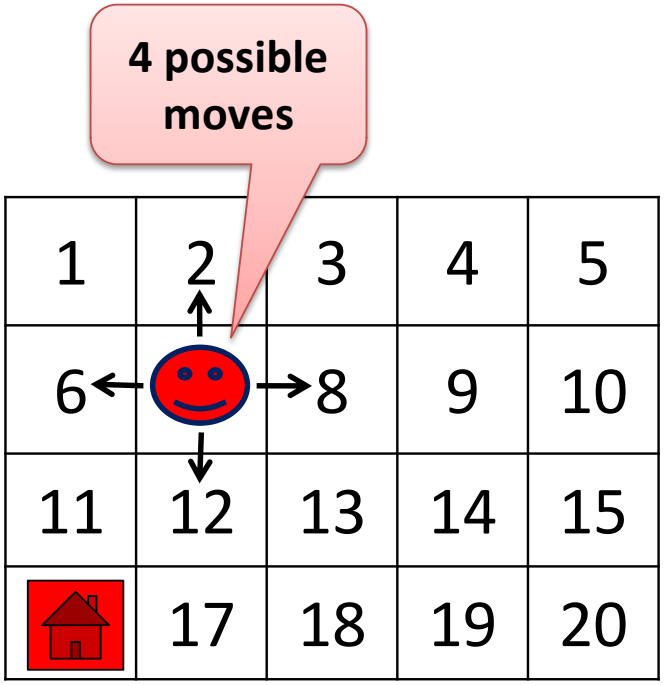


- Step 1: Plan **blue**
  - Done!
- Step 2: Plan **red**
  - **Done!**
- **…**
- **Step N: Plan Nth agent**

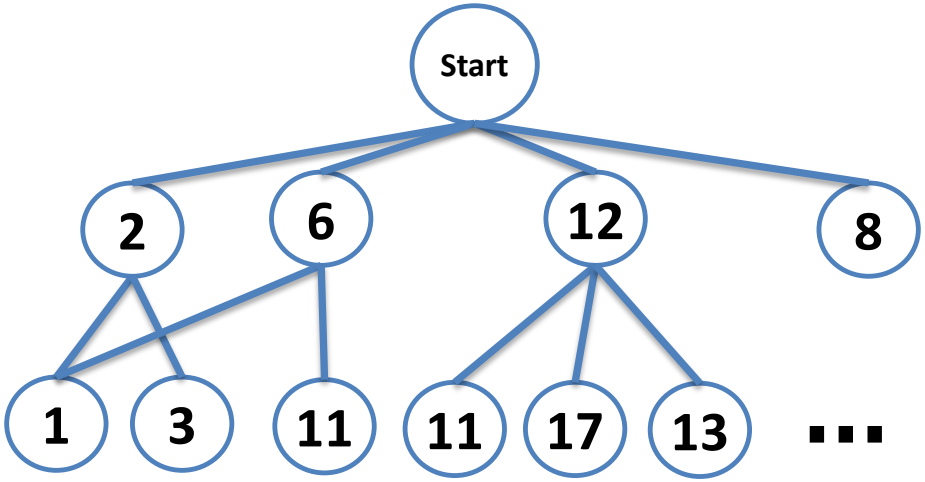


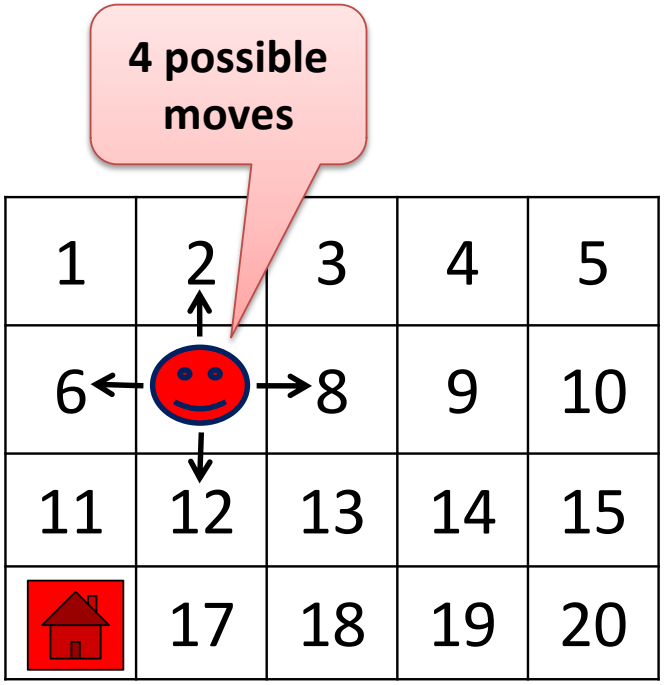
Prioritized Planning (Silver 2005) Analysis: First Agent





Prioritized Planning (Silver 2005) Analysis: First Agent

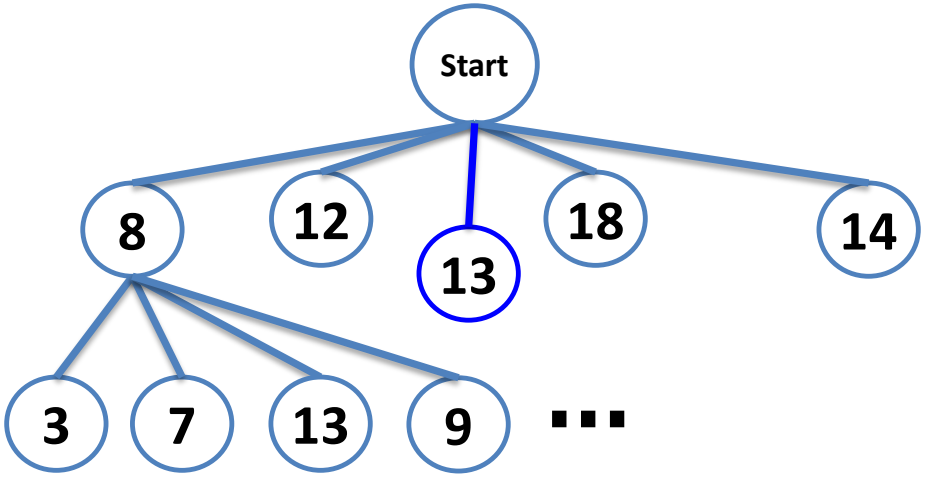


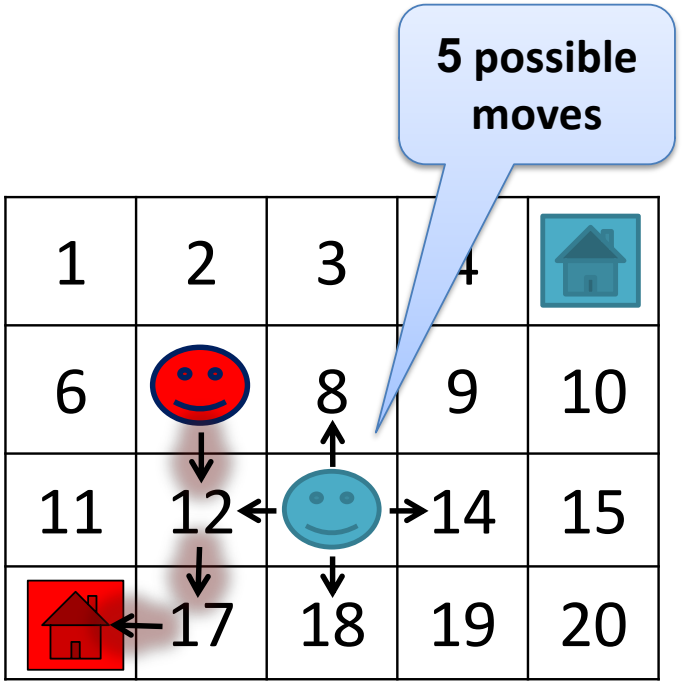


Singe-agent pathfinding

- A state is the agent's location
- Number of states = 4 x 5
- Branching factor = 4

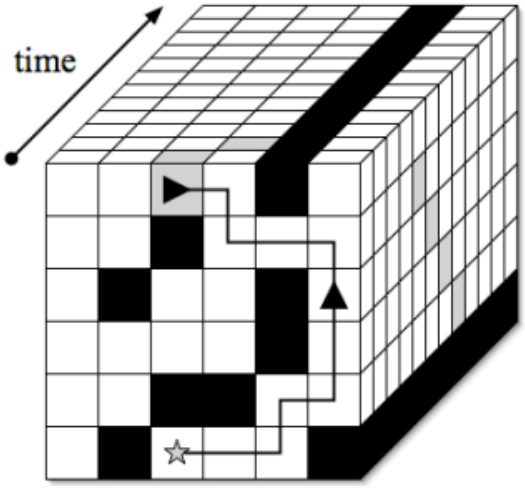
Prioritized Planning (Silver 2005) Analysis: Second Agent

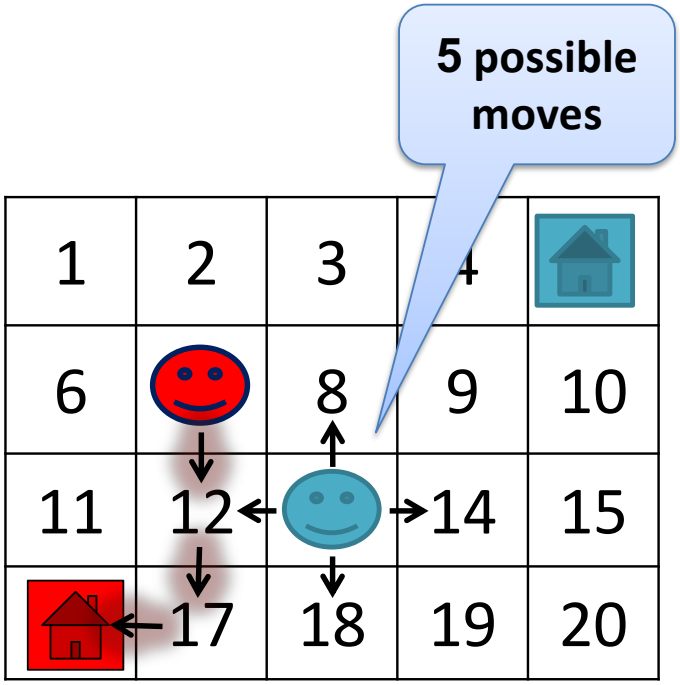




- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

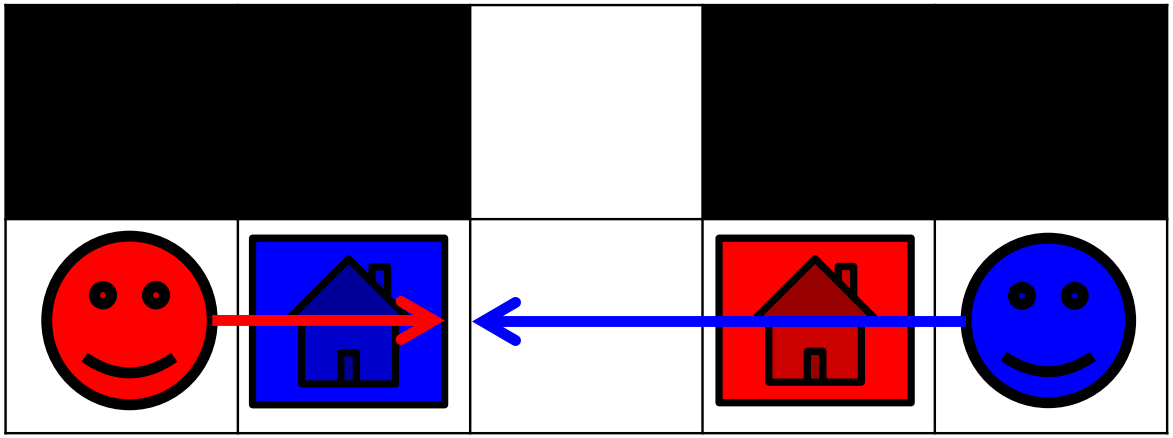
Prioritized Planning (Silver 2005) Analysis: Second Agent



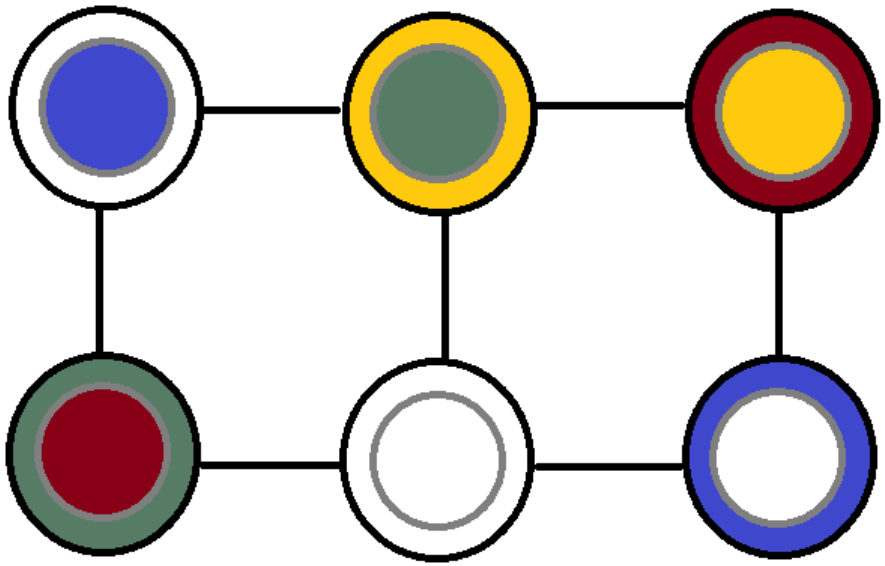


- A state is a (location,time) pair
- Number of states = 4 x 5 x maxTime
- Branching factor = 4+1

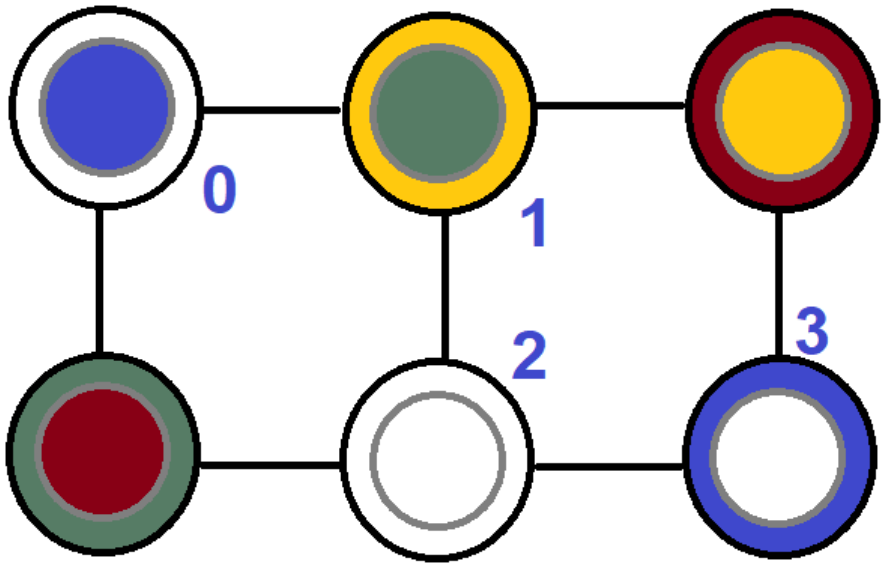
- Complexity?
  - **Polynomial** in the grid size and max time
- Soundness?
  - **Yes!**
- Complete? Optimal?
  - **No** ☹



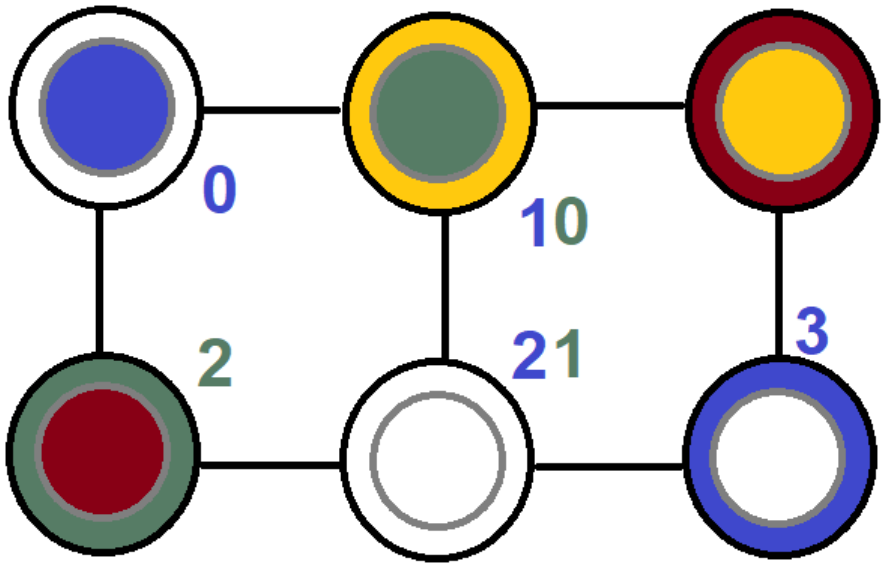
Prioritized planning (cooperative A*) example



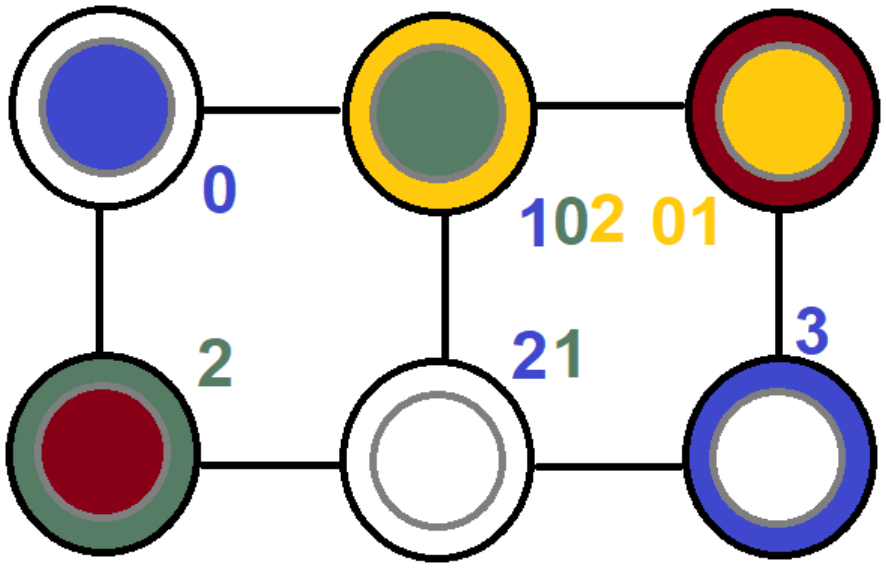
Prioritized planning (cooperative A*) example



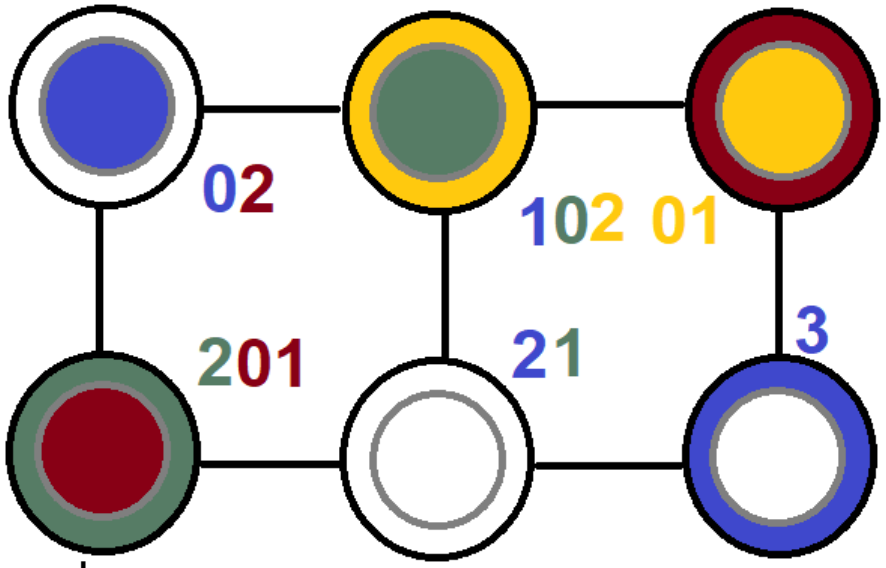
Prioritized planning (cooperative A*) example



Prioritized planning (cooperative A*) example



Prioritized planning (cooperative A*) example



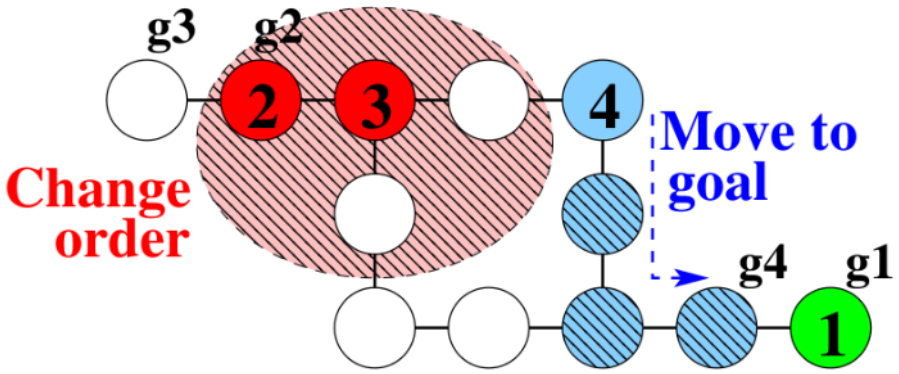
Impossible: red cannot move!

We find a solution if we order the agents as red, green, blu and yellow, but the algorithm does not guarantee to find it for *any* ordering

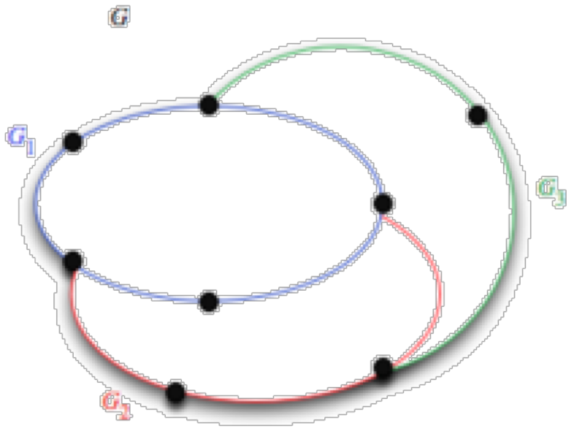
- Smart agent **prioritization**
  - Conflict oriented WHCA* [Bnaya and Felner '14]
  - Re-prioritization and safe intervals [Andreychuk and Yakovlev '18]
- Integrate **planning and execution**
  - Windowed Hierarchical CA* [Silver '06]

	Suboptimal	**Optimal**
Incomplete	• Cooperative A* • WHCA*	?
Complete	?	?

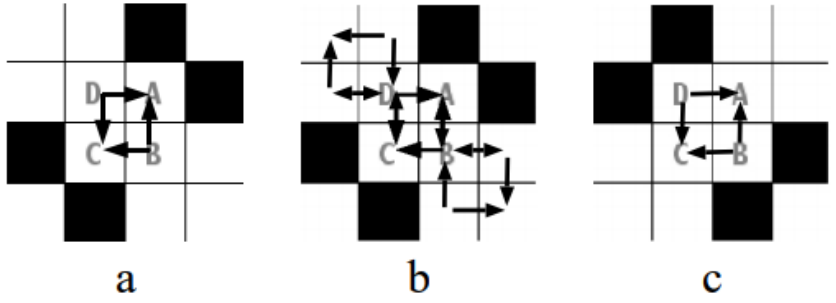
Examples



Push and Swap (Luna and Bekris '13)



Bibox (Surynek '09)



FAR (Wang and Botea '08)

	Suboptimal	**Optimal**
Incomplete	• Cooperative A* • WHCA*	?
Complete	• Kornhauser et al. ’84 • Push & Swap (Luna & Bekris) • Bibox (Surynek) …	?

Increasing Cost Tree Search (Sharon et al. '12)

High-level

Is there a solution with costs

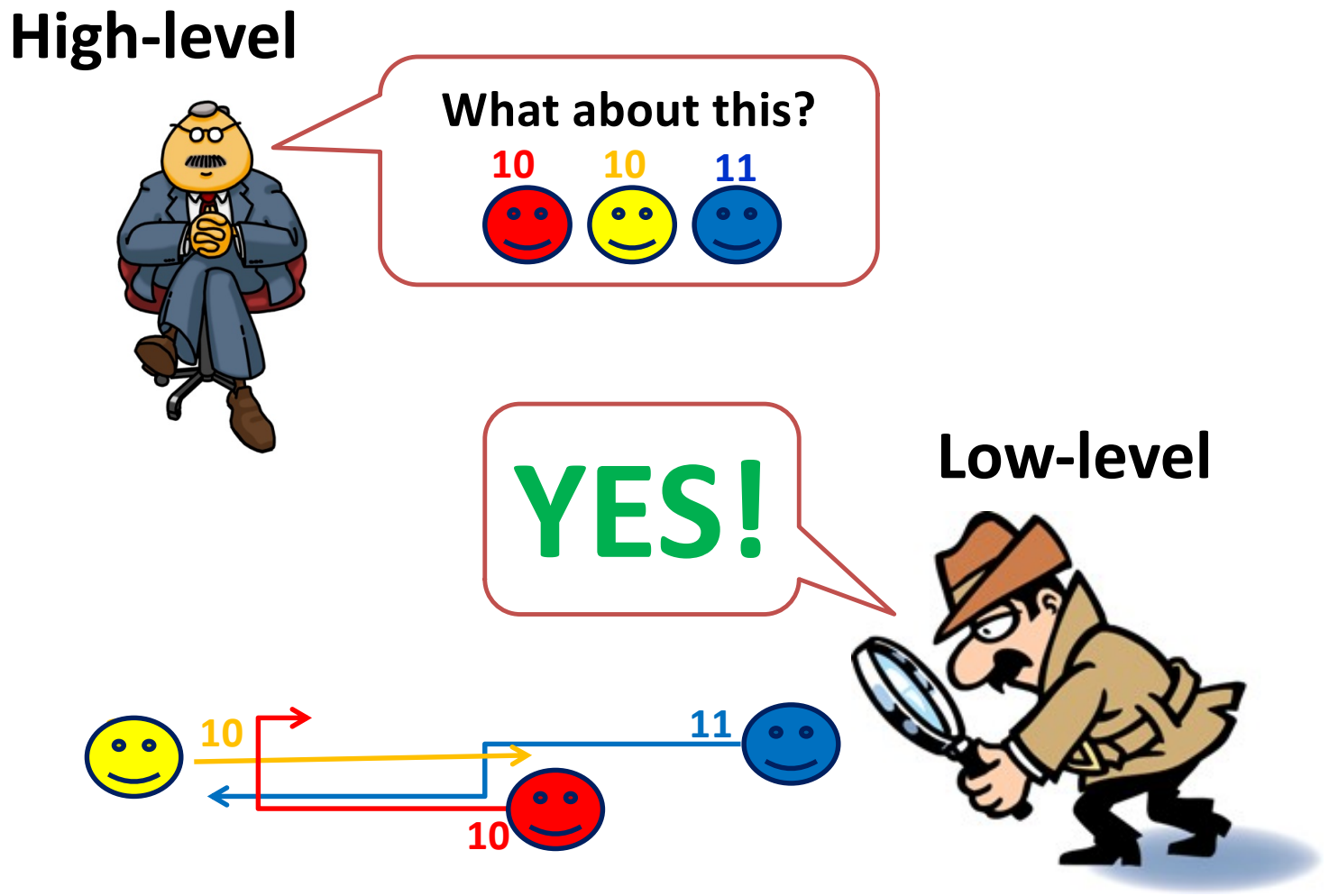
10 10 10

?

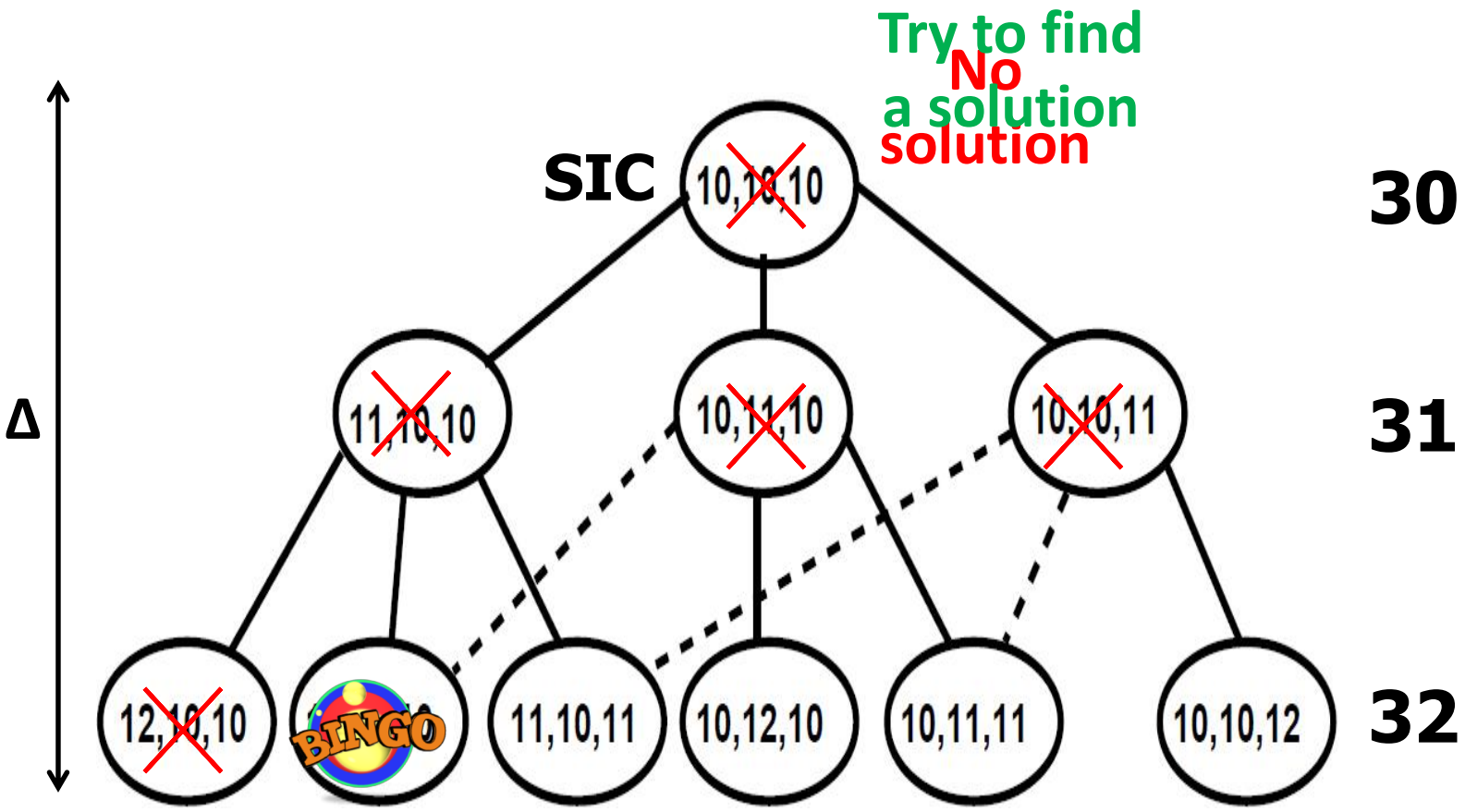
NO!

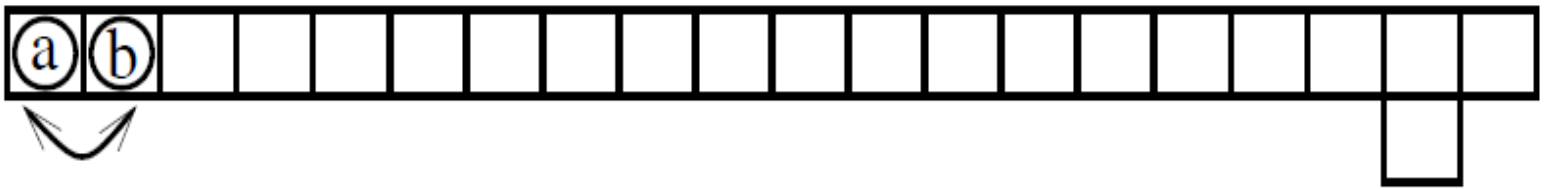
Low-level

Increasing Cost Tree Search (Sharon et al. '12)

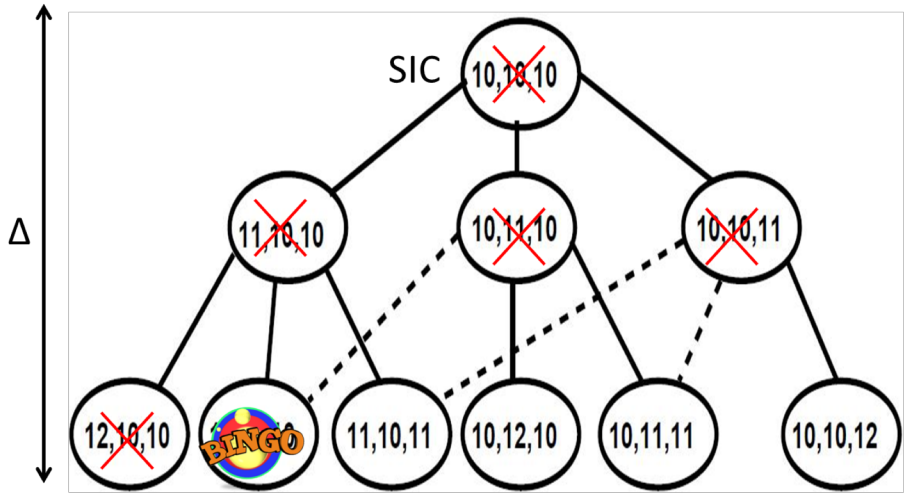


Increasing Cost Tree Search (Sharon et al. '12)

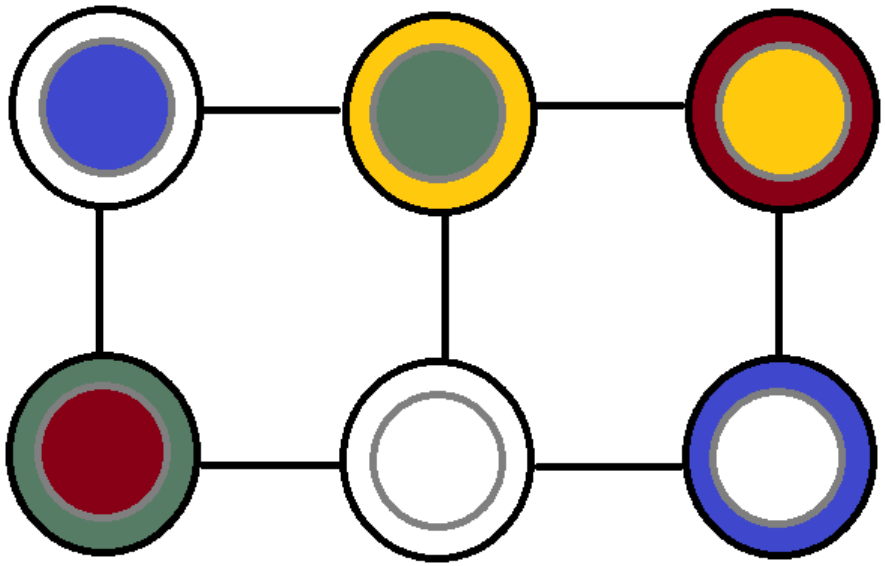




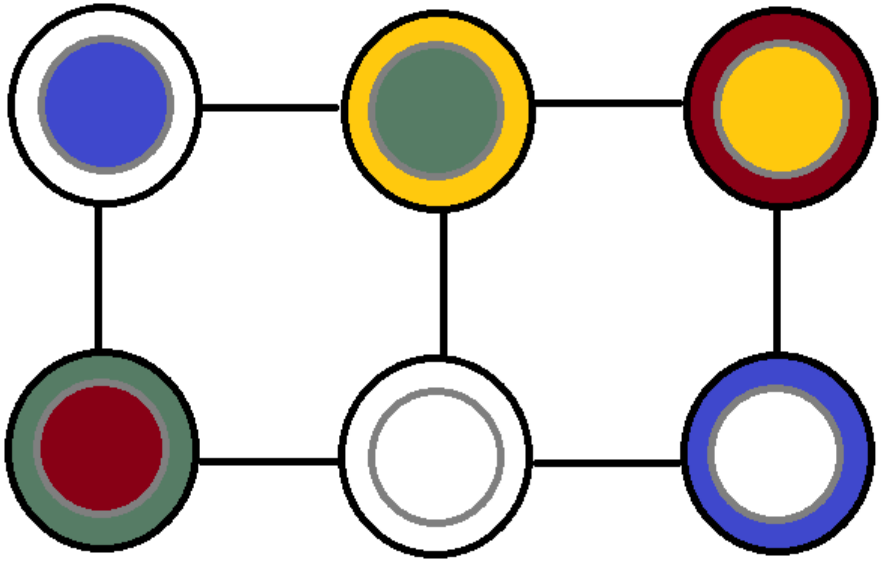
- A*: solved in **51**ms
- **ICTS Complexity depends on Δ**
  - Sum of single agent costs =2 **BUT** optimal solution =74



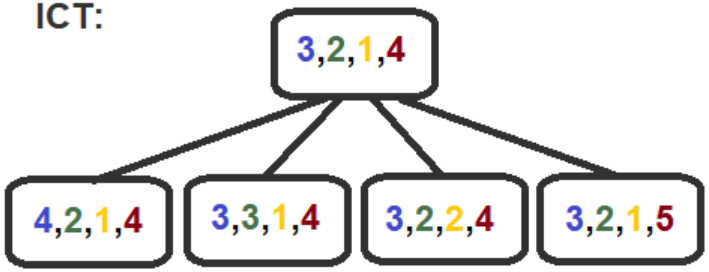
Increasing Cost Search Tree (ICTS) example



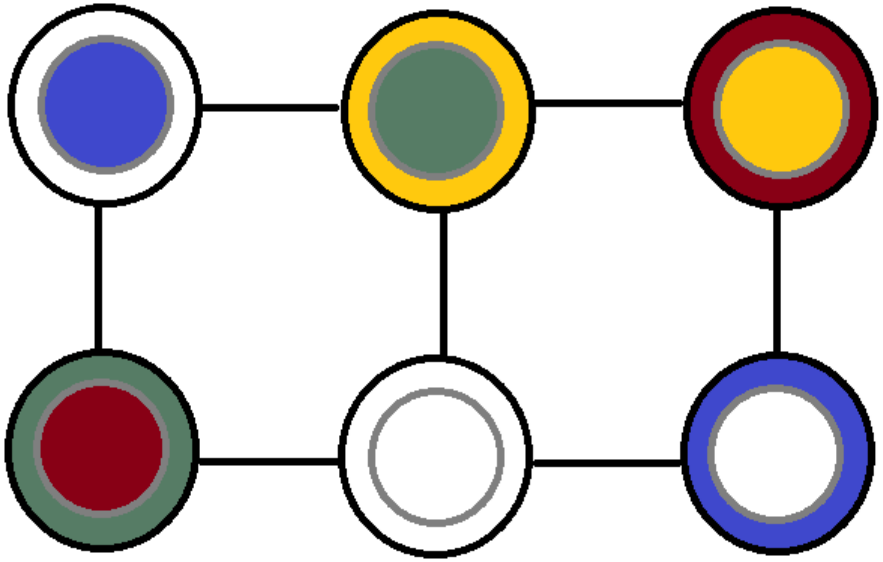
Increasing Cost Tree Search (ICTS) example



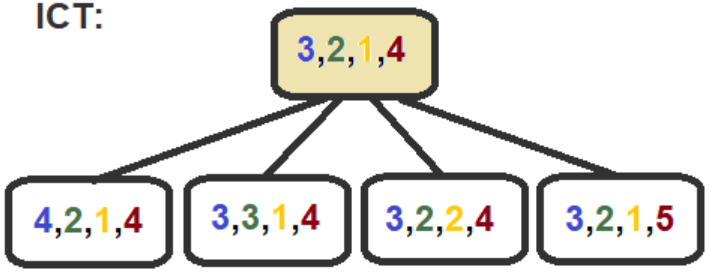
ICT:



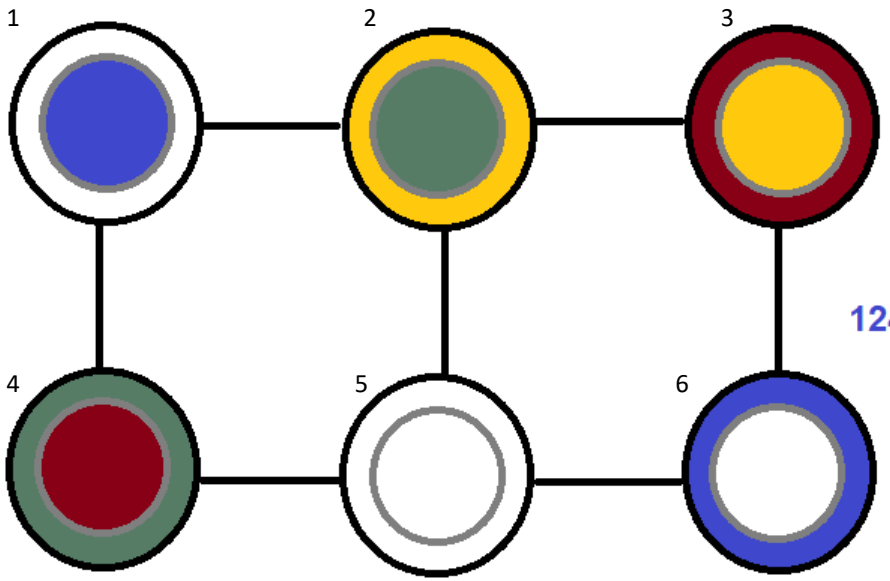
Increasing Cost Tree Search (ICTS) example



ICT:



Increasing Cost Tree Search (ICTS) example



MDD^3 MDD^2 MDD^1 MDD^3

MDD^3 da BFS

1

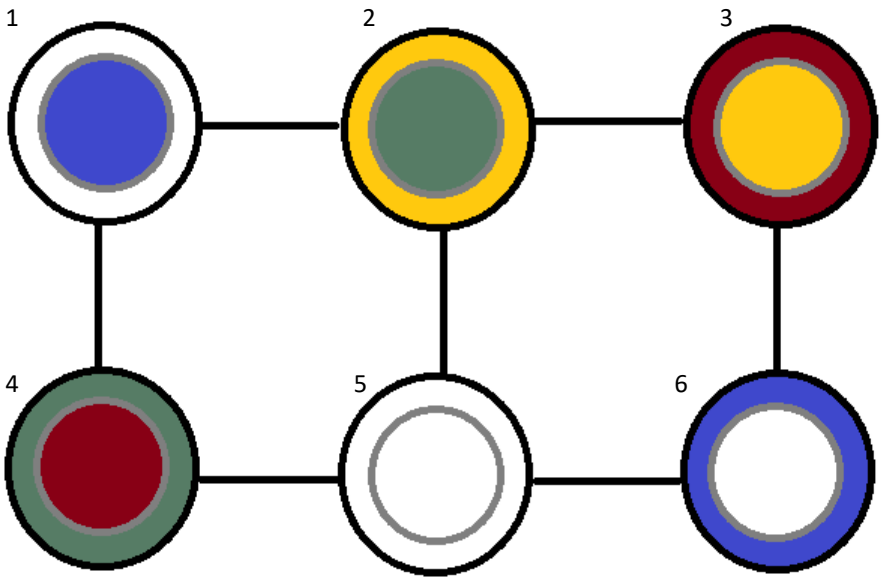
1 2 4

1 2 4 1 2 3 5 1 4 5

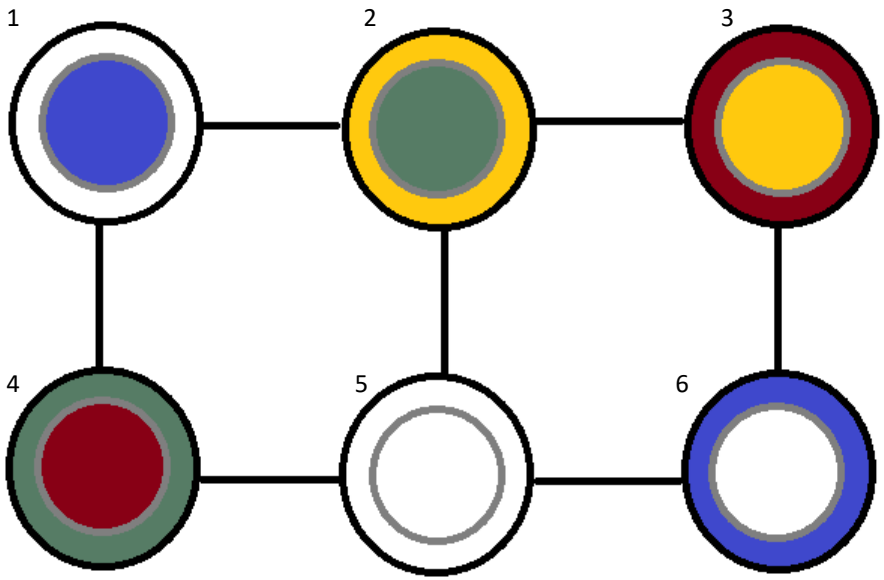
124 1235 145 124 1235 236 2456 124 145 2456

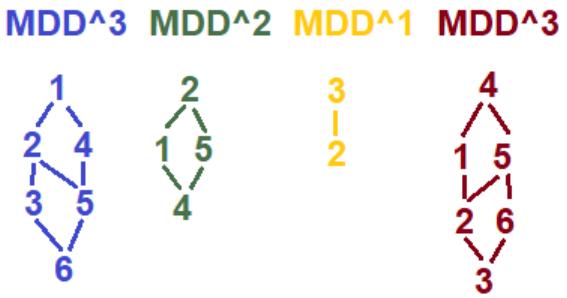
MDD: Multi-value Decision Diagram

Increasing Cost Tree Search (ICTS) example

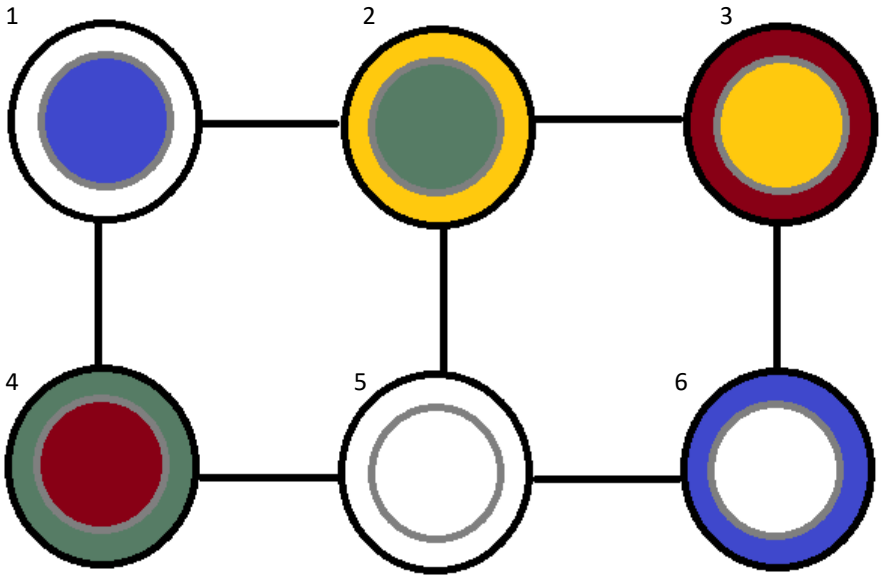


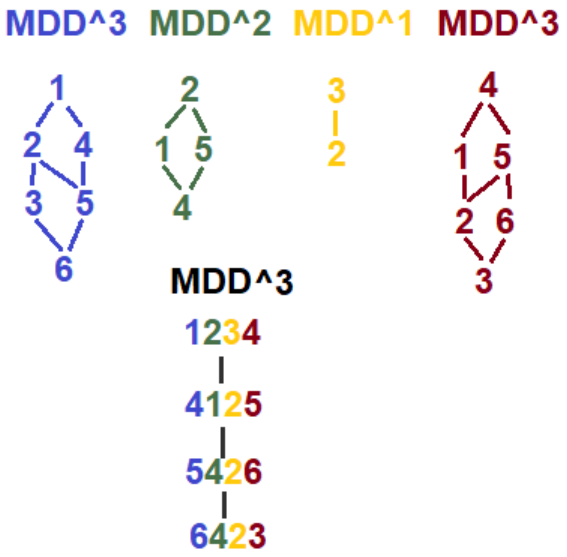
Increasing Cost Tree Search (ICTS) example





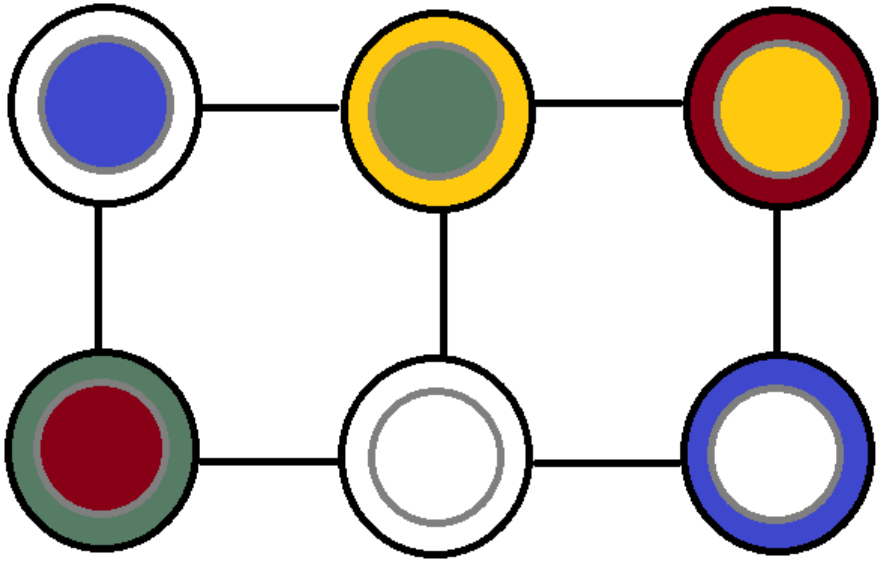
Increasing Cost Tree Search (ICTS) example

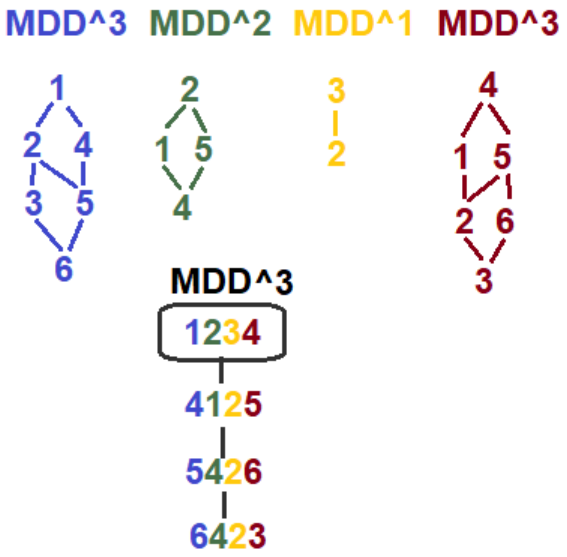




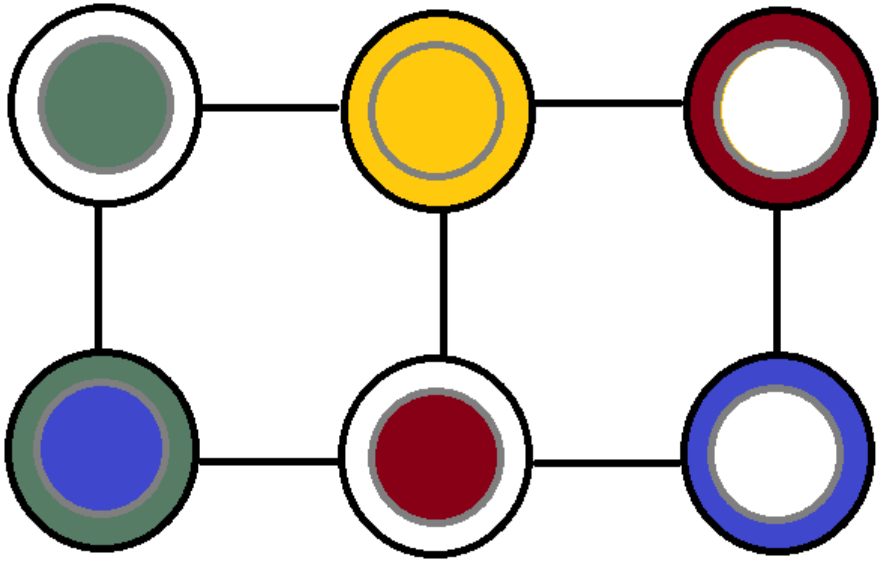
Solution found!

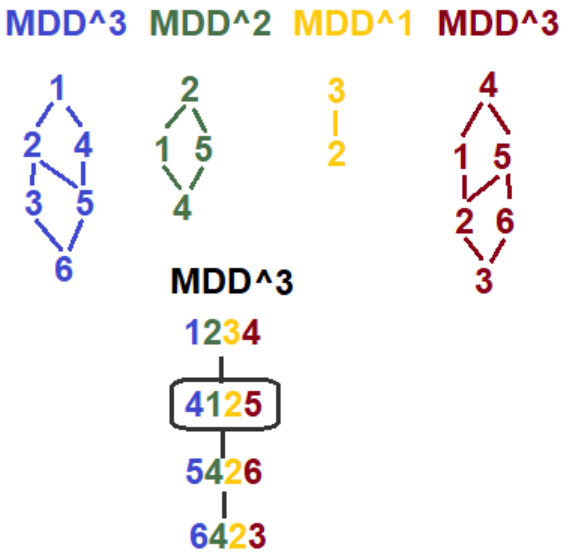
Increasing Cost Tree Search (ICTS) example



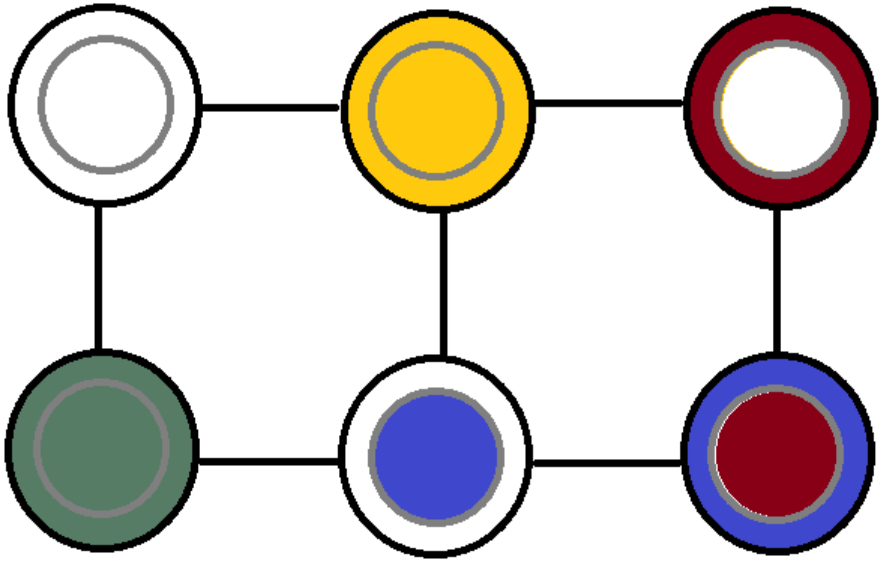


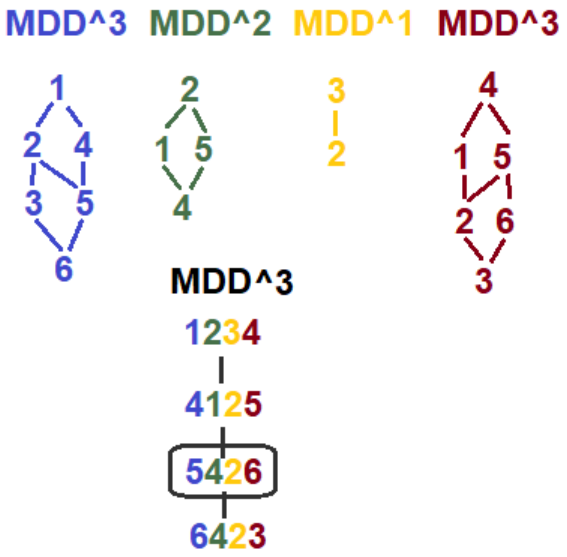
Increasing Cost Tree Search (ICTS) example: execution



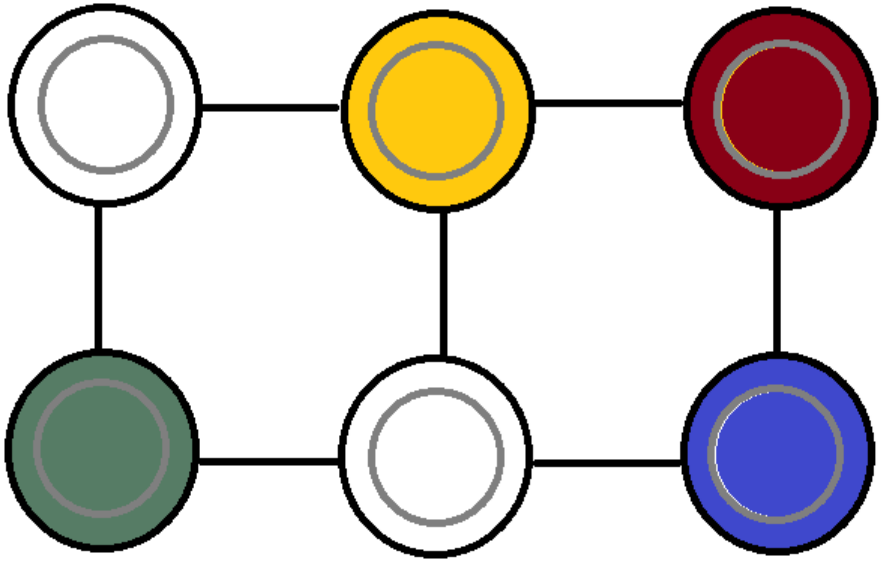


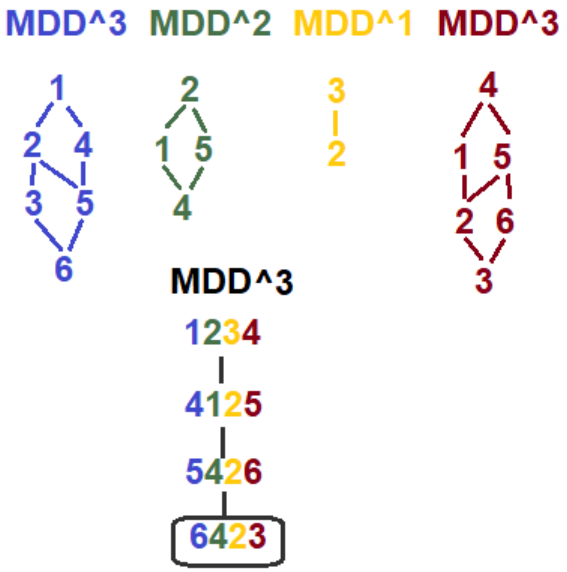
Increasing Cost Tree Search (ICTS) example: execution





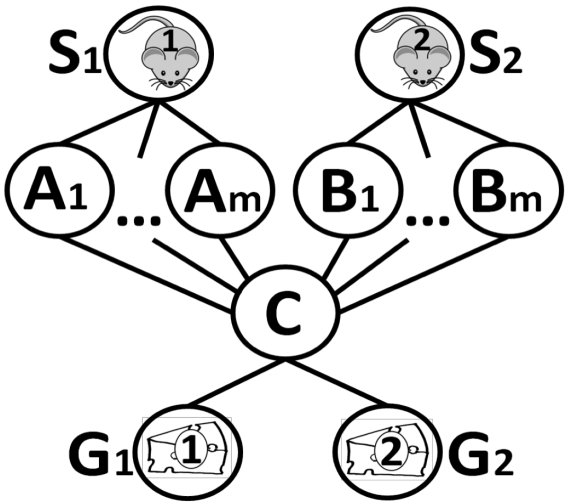
Increasing Cost Tree Search (ICTS) example: execution





CBS only performs single agents

But, many times, and under different constraints



Conflict: agent 1 and agent 2 plan to occupy C at time 2

Constrain agent 1, avoid C at time 2
or
Constrain agent 2 to avoid C at time 2

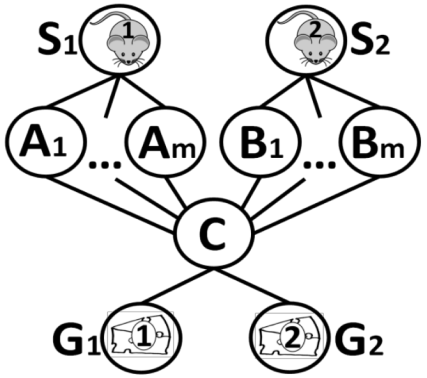
The Constraint Tree

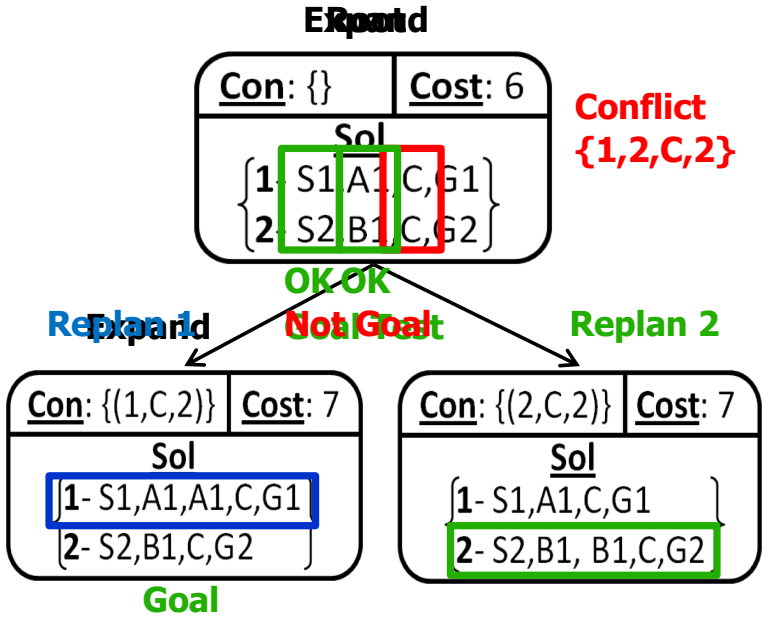
Nodes:

- A set of individual constraints for each agent
- A set of paths consistent with the constraints

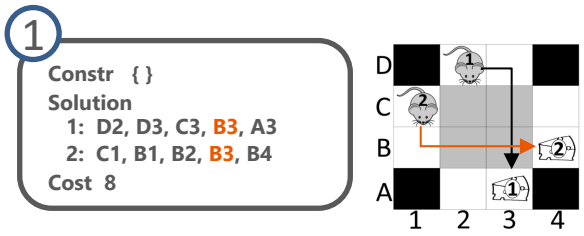
Goal test:

- Are the paths conflict free





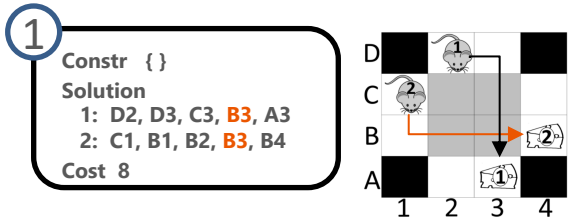
Conflict-Based Search (CBS) example



Algorithm 1: high-level of CBS

Input: MAPF instance
1 $R.constraints = \emptyset$
2 $R.solution$ = find individual paths using the low-level()
3 $R.cost = SIC(R.solution)$
4 insert R to OPEN
5 **while** OPEN *not empty* **do**
6 P ← best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** *P has no conflict* **then**
9 **return** P.solution // *P is goal*
10 C ← first conflict (a_i, a_j, v, t) in P
11 **foreach** *agent* a_i *in C* **do**
12 A ← new node
13 A.constriants ← P.constriants + (a_i, s, t)
14 A.solution ← P.solution.
15 Update A.solution by invoking low-level(a_i)
16 $A.cost = SIC(A.solution)$
17 Insert A to OPEN

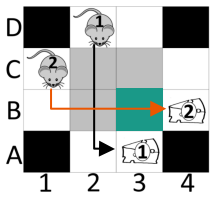
Conflict-Based Search (CBS) example



Algorithm 1: high-level of CBS

Input: MAPF instance
1 $R.constraints = \emptyset$
2 $R.solution$ = find individual paths using the low-level()
3 $R.cost = SIC(R.solution)$
4 insert R to OPEN
5 **while** OPEN *not empty* **do**
6 P ← best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** *P has no conflict* **then**
9 **return** P.solution // *P is goal*
10 C ← first conflict (a_i, a_j, v, t) in P
11 **foreach** *agent* a_i *in C* **do**
12 A ← new node
13 A.constriants ← P.constriants + (a_i, s, t)
14 A.solution ← P.solution.
15 Update A.solution by invoking low-level(a_i)
16 $A.cost = SIC(A.solution)$
17 Insert A to OPEN

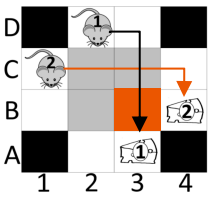
Conflict-Based Search (CBS) example



2
Constr (1,B3,3)
Solution
1: D2, C2, B2, A2, A3
2: C1, B1, B2, B3, B4
Cost 8

1
Constr { }
Solution
1: D2, D3, C3, B3, A3
2: C1, B1, B2, B3, B4
Cost 8

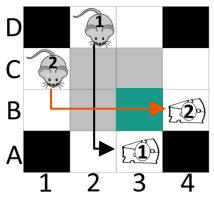
3
Constr (2,B3,3)
Solution
1: D2, D3, C3, B3, A3
2: C1, C2, C3, C4, B4
Cost 8



Algorithm 1: high-level of CBS

Input: MAPF instance
1 $R.constraints = \emptyset$
2 $R.solution$ = find individual paths using the low-level()
3 $R.cost = SIC(R.solution)$
4 insert R to OPEN
5 **while** OPEN *not empty* **do**
6 P ← best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** *P has no conflict* **then**
9 **return** P.solution // *P is goal*
10 C ← first conflict (a_i, a_j, v, t) in P
11 **foreach** *agent* a_i *in C* **do**
12 A ← new node
13 A.constriants ← P.constriants + (a_i, s, t)
14 A.solution ← P.solution.
15 Update A.solution by invoking low-level(a_i)
16 $A.cost = SIC(A.solution)$
17 Insert A to OPEN

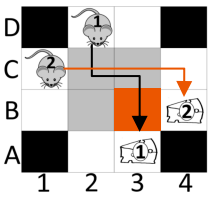
Conflict-Based Search (CBS) example



2
Constr (1,B3,3)
Solution
1: D2, C2, B2, A2, A3
2: C1, B1, B2, B3, B4
Cost 8

1
Constr { }
Solution
1: D2, D3, C3, B3, A3
2: C1, B1, B2, B3, B4
Cost 8

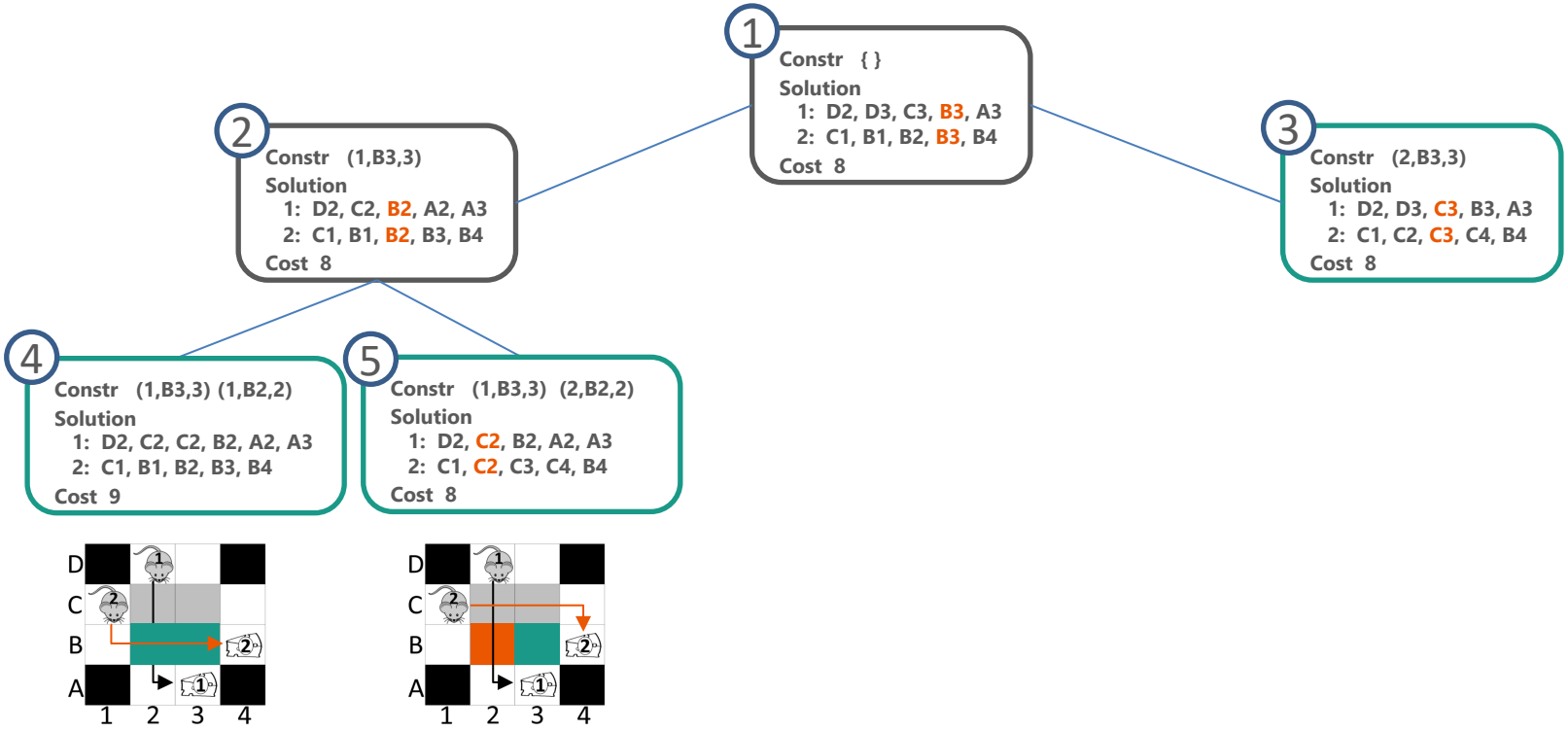
3
Constr (2,B3,3)
Solution
1: D2, D3, C3, B3, A3
2: C1, C2, C3, C4, B4
Cost 8



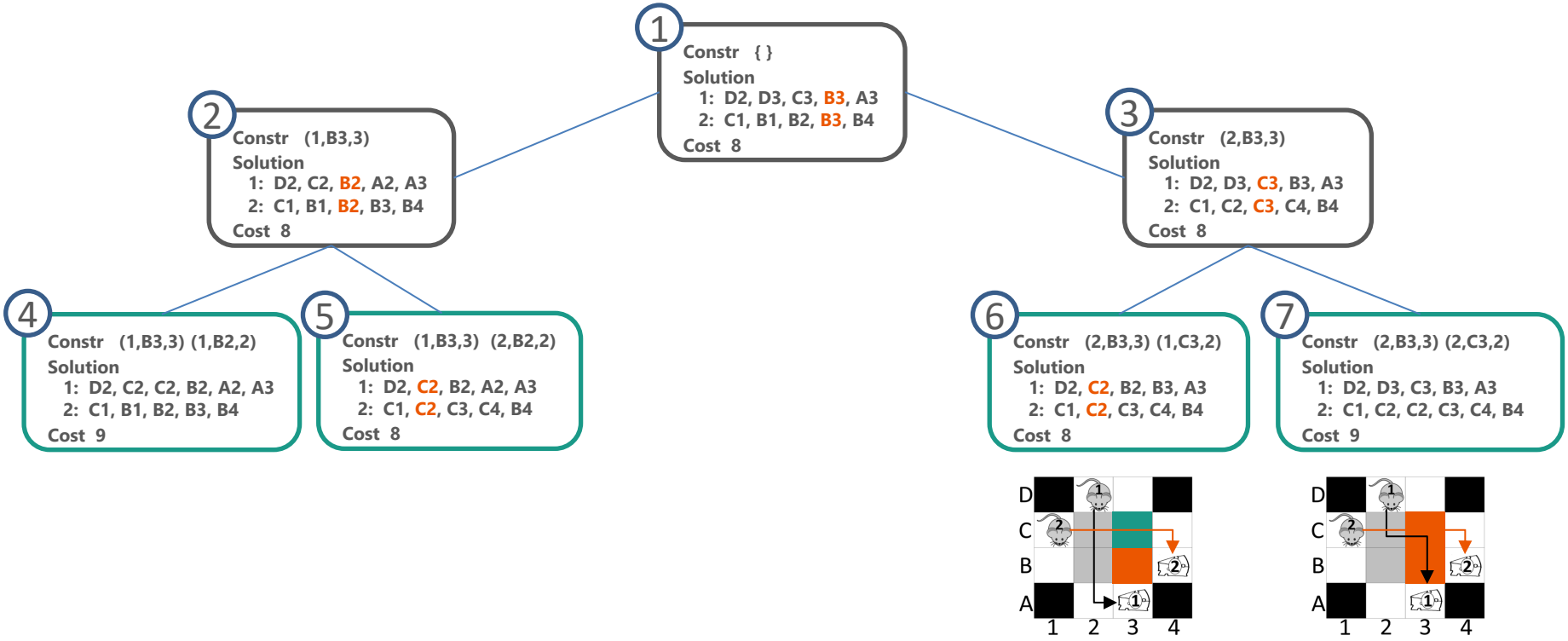
Algorithm 1: high-level of CBS

Input: MAPF instance
1 $R.constraints = \emptyset$
2 $R.solution$ = find individual paths using the low-level()
3 $R.cost = SIC(R.solution)$
4 insert R to OPEN
5 **while** OPEN *not empty* **do**
6 P ← best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** *P has no conflict* **then**
9 **return** P.solution // *P is goal*
10 C ← first conflict (a_i, a_j, v, t) in P
11 **foreach** *agent* a_i *in C* **do**
12 A ← new node
13 A.constriants ← P.constriants + (a_i, s, t)
14 A.solution ← P.solution.
15 Update A.solution by invoking low-level(a_i)
16 $A.cost = SIC(A.solution)$
17 Insert A to OPEN

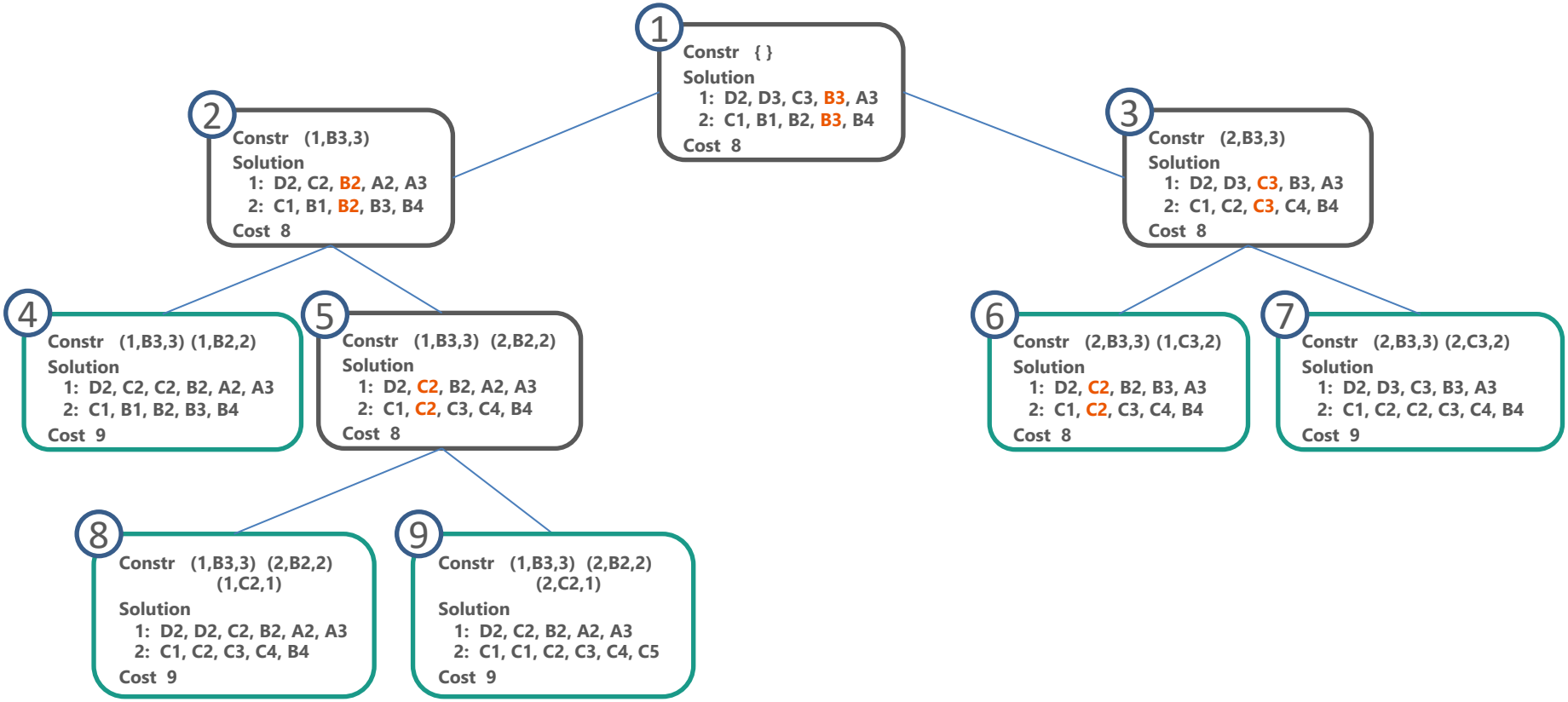
Conflict-Based Search (CBS) example



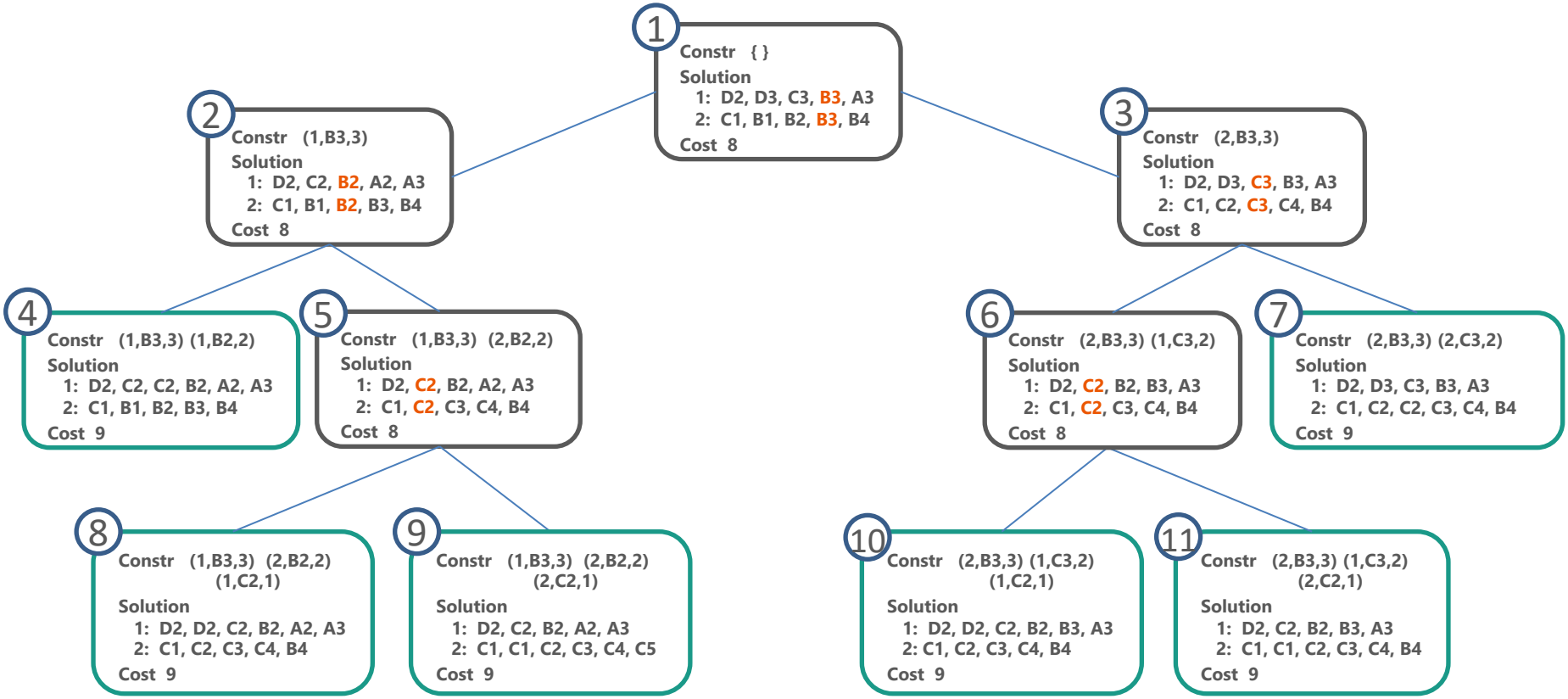
Conflict-Based Search (CBS) example



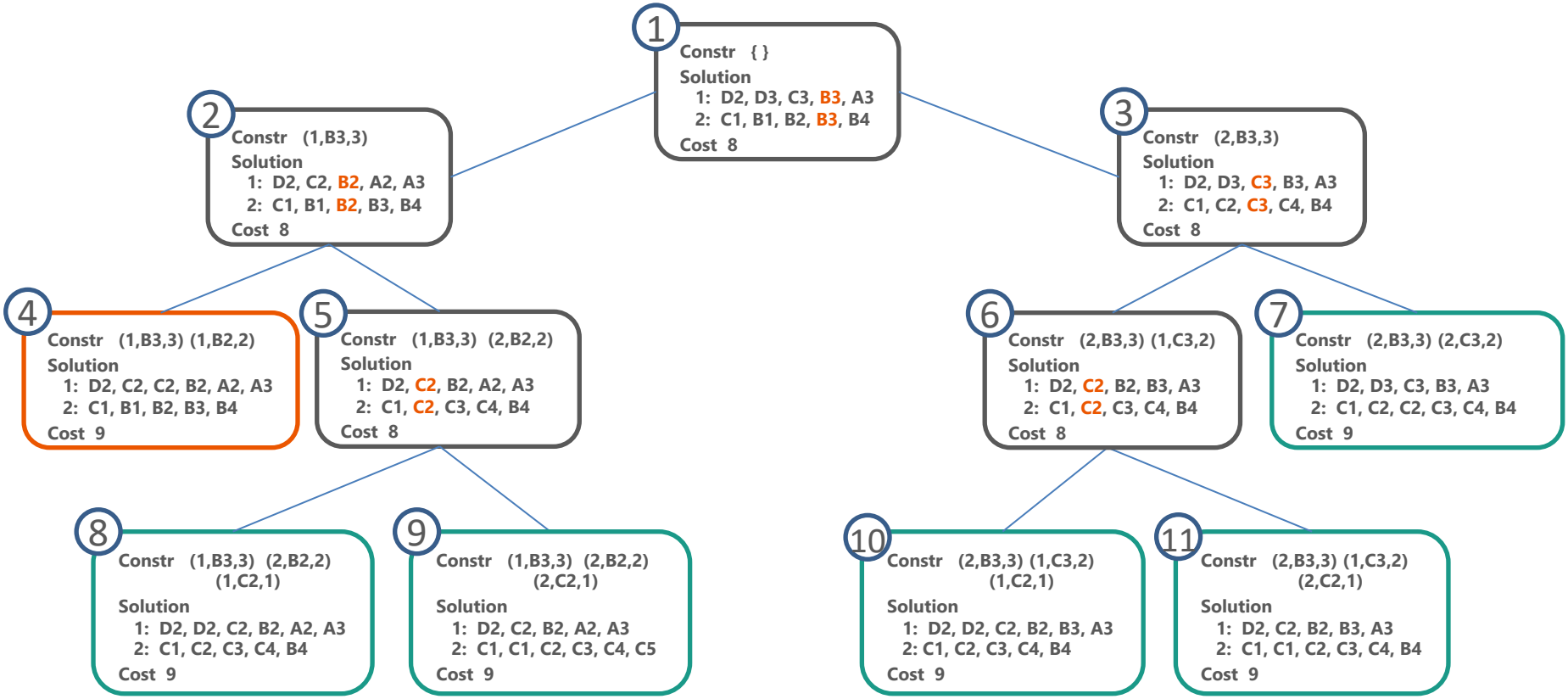
Conflict-Based Search (CBS) example



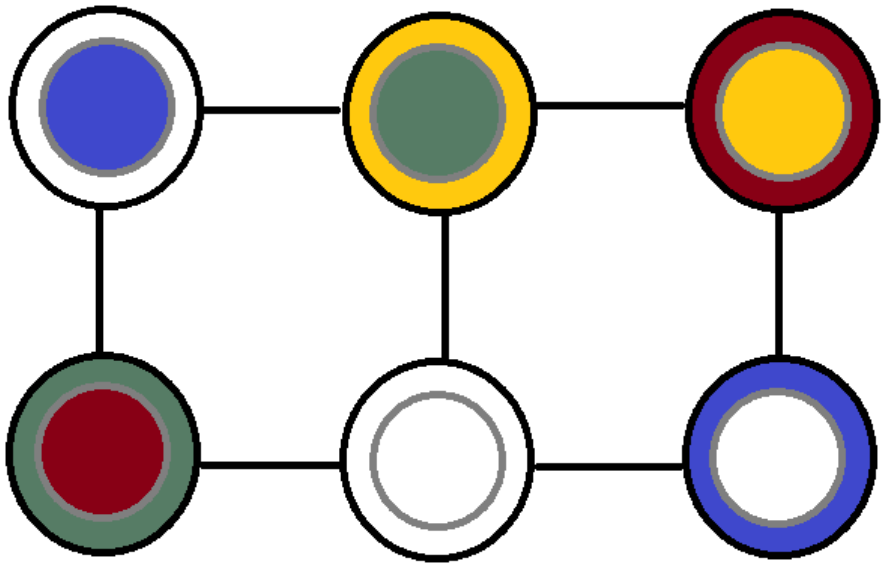
Conflict-Based Search (CBS) example



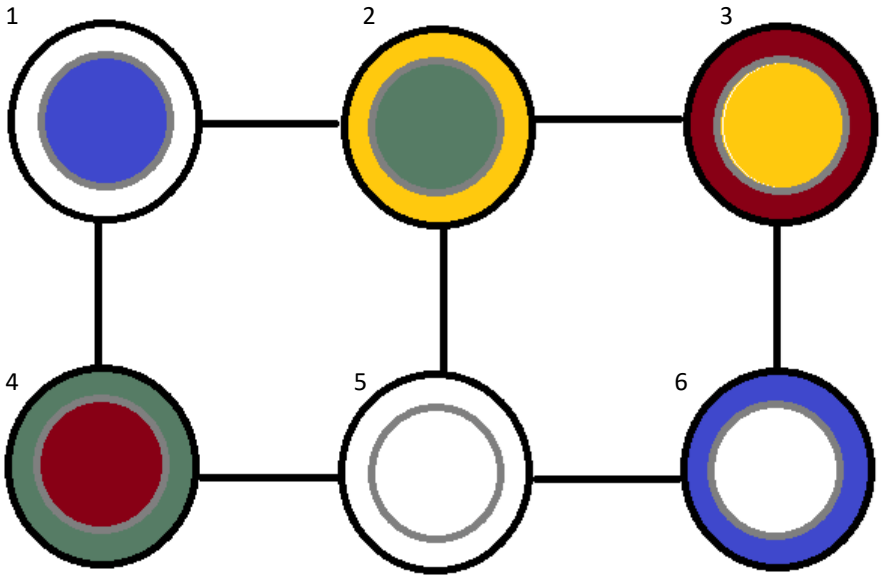
Conflict-Based Search (CBS) example



Conflict-Based Search (CBS): another example



Conflict-Based Search (CBS): another example

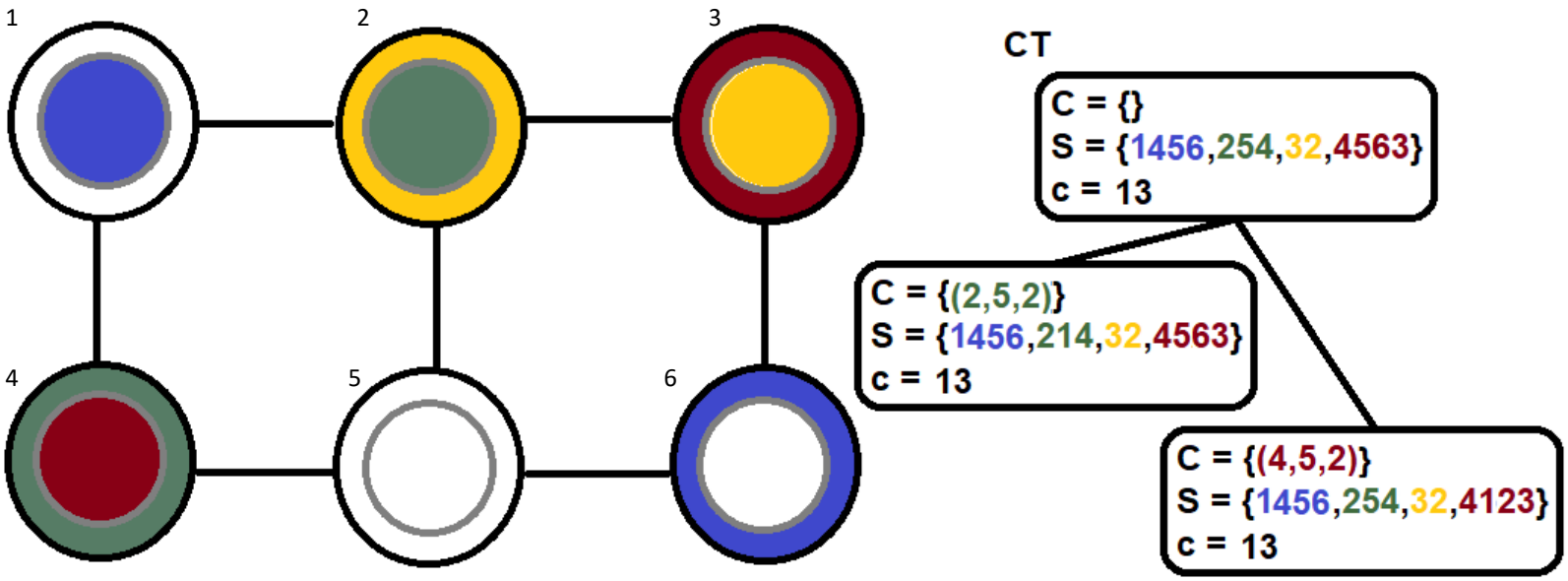


CT

C = {}
S = {1456,254,32,4563}
c = 13

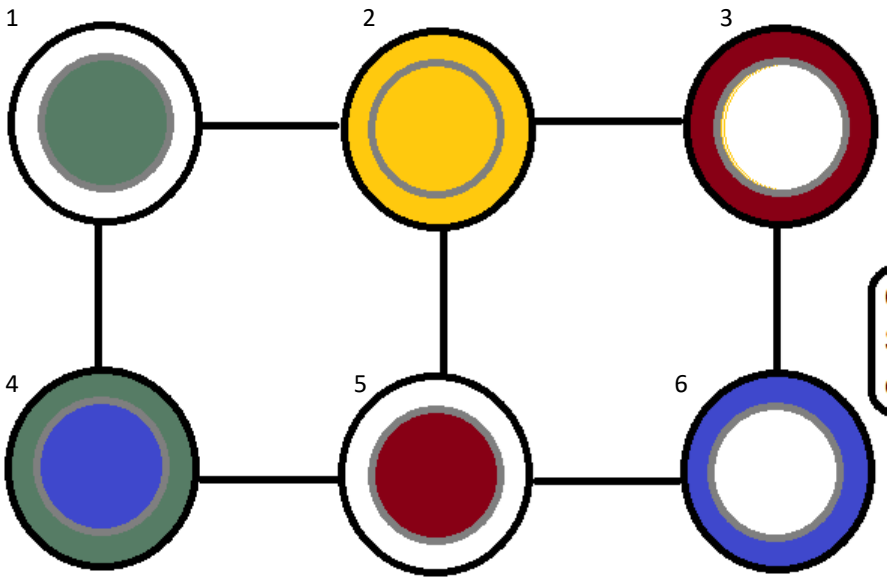
The solution is not validated: conflict at node 5 at time 2 between green and red agents

Conflict-Based Search (CBS): another example



Left child: solution is validated

Conflict-Based Search (CBS): another example, execution



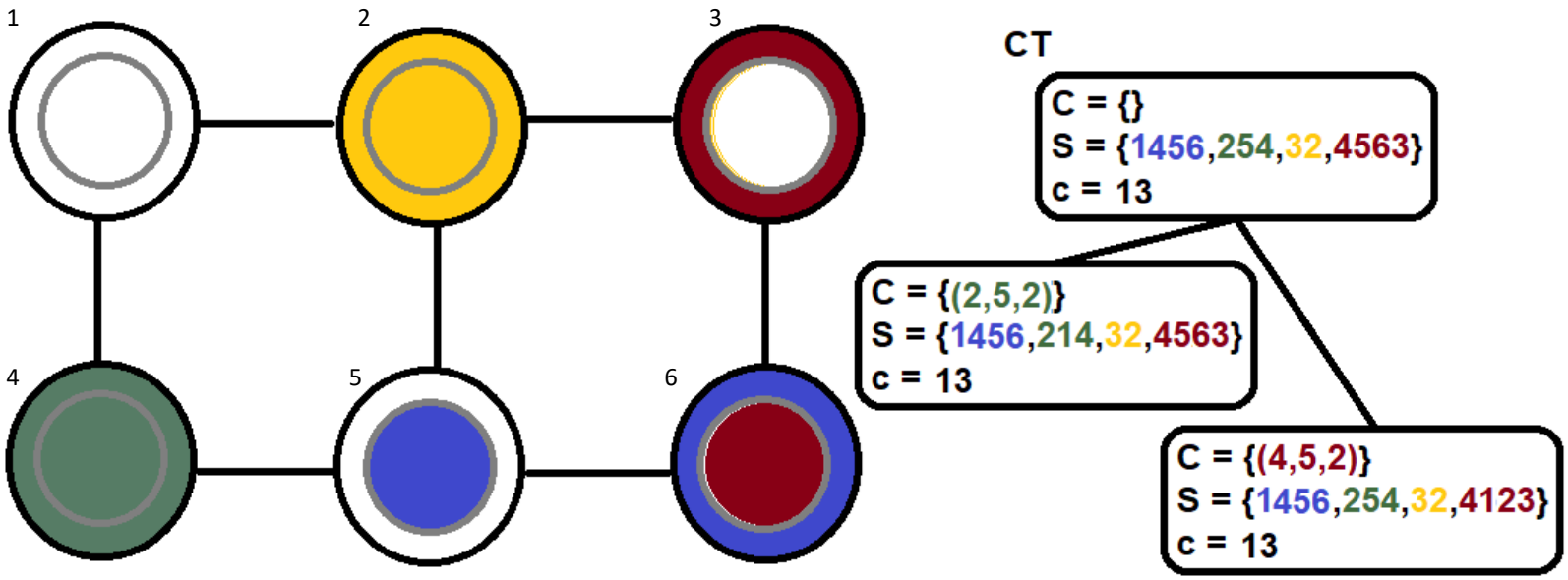
CT

C = {}
S = {1456,254,32,4563}
c = 13

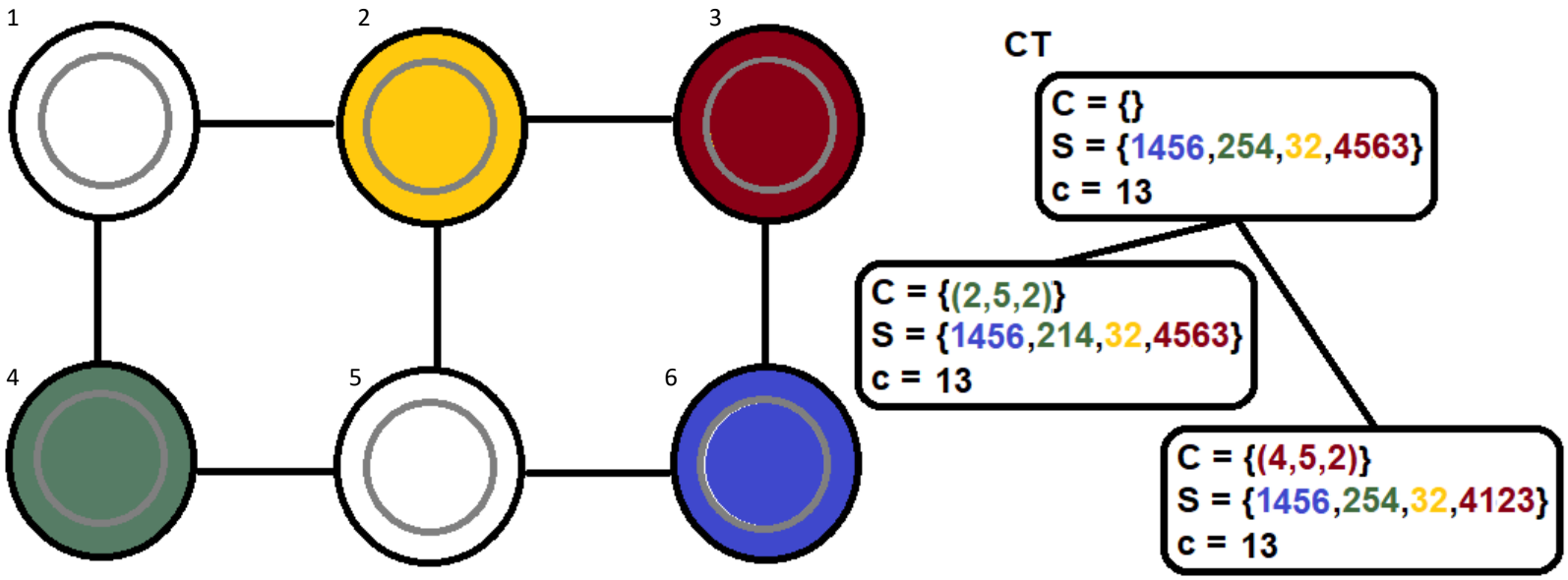
C = {(2,5,2)}
S = {1456,214,32,4563}
c = 13

C = {(4,5,2)}
S = {1456,254,32,4123}
c = 13

Conflict-Based Search (CBS): another example, execution

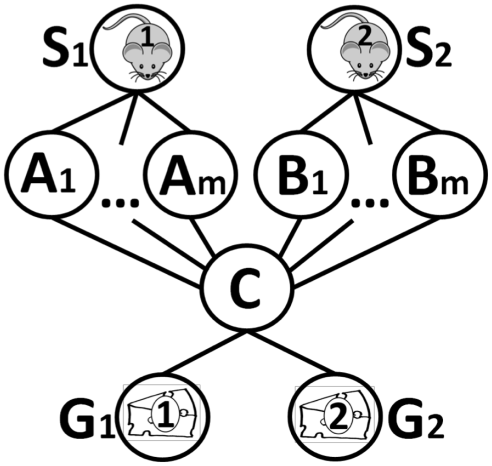


Conflict-Based Search (CBS): another example, execution



Analysis: Example 1

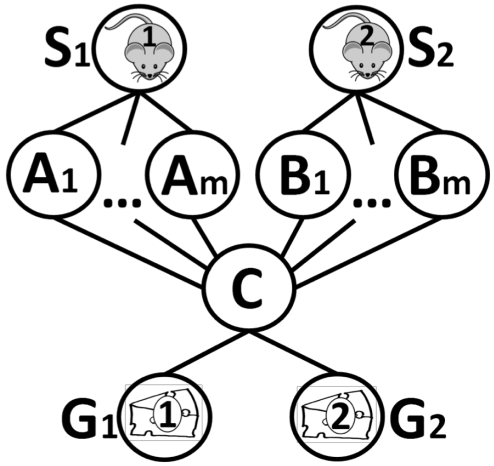
- How many states **A*** will expand?
- How many states **CBS** will?



Analysis: Example 1

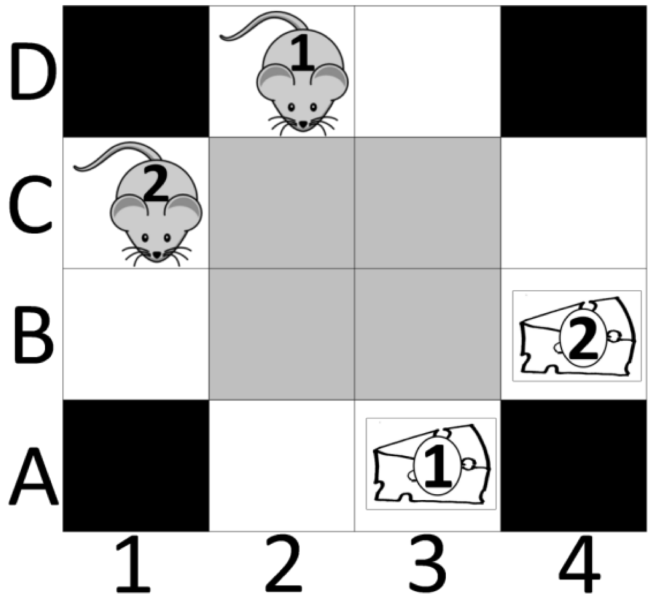
- **A*** : m^2+3 = $O(m^2)$ states
- **CBS:** $2m+14$ = $O(m)$ states

When m > 4 CBS will examine fewer states than A*

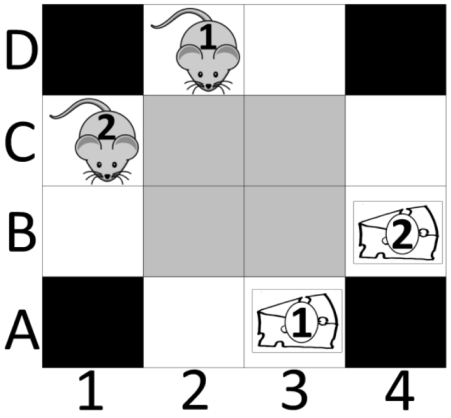


Analysis: Example 2

- States expanded by CBS**?**
- States expanded by A***?**



Analysis: Example 2



- 4 optimal solutions for each agent
- Each pair of solutions has a conflict

- Rough analysis:
  - **CBS:** exponential in #conflicts = 54 states
  - **A*:** exponential in #agents = 8 states

Trends observed
- **In open spaces: use A***
- **In bottlenecks: use CBS**

What if I have both?

CBS Enhancements

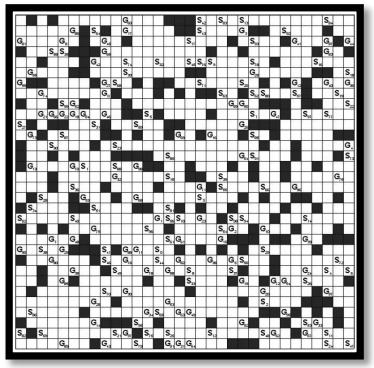
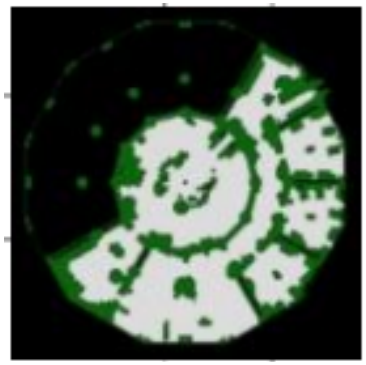
- Which conflict to resolve? [Boyarski et al. '16]
- What to do after merging? [Boyarski et al. '16]
- Heuristics for the constraint tree search [Felner et al. '18]
- Augmenting CBS with human knowledge [Cohen et al.]
- Which low-level solver to use?
- When to merge the agents ?

…

Summary – No Universal Winner

- A* (M*, EPEA*, A*+OD+ID)
  - Main factors: #agents, graph size, heuristic accuracy
- ICTS
  - Main factors: #agents, Δ, graph size
- CBS and its variants
  - Main factors: #conflicts

Where to use what?

	Suboptimal	**Optimal**
Incomplete	• Cooperative A* • WHCA*	?
Complete	• Kornhauser et al. ’84 • Push & Swap (Luna & Bekris) • Bibox (Surynek) …	• A*+OD+ID (Standley) • ICTS (Sharon et al.) • M* (Wagner & Choset) • CBS (Sharon et al.) …

An algorithm is bounded suboptimal iff

- **It accepts a parameter ϵ**
- **It outputs a solution whose cost is at most $(1+\epsilon)\cdot$Optimal**

How to create a bounded suboptimal algorithm?

- Different search algorithms
- Inadmissible heuristics

Optional: software tools (1)

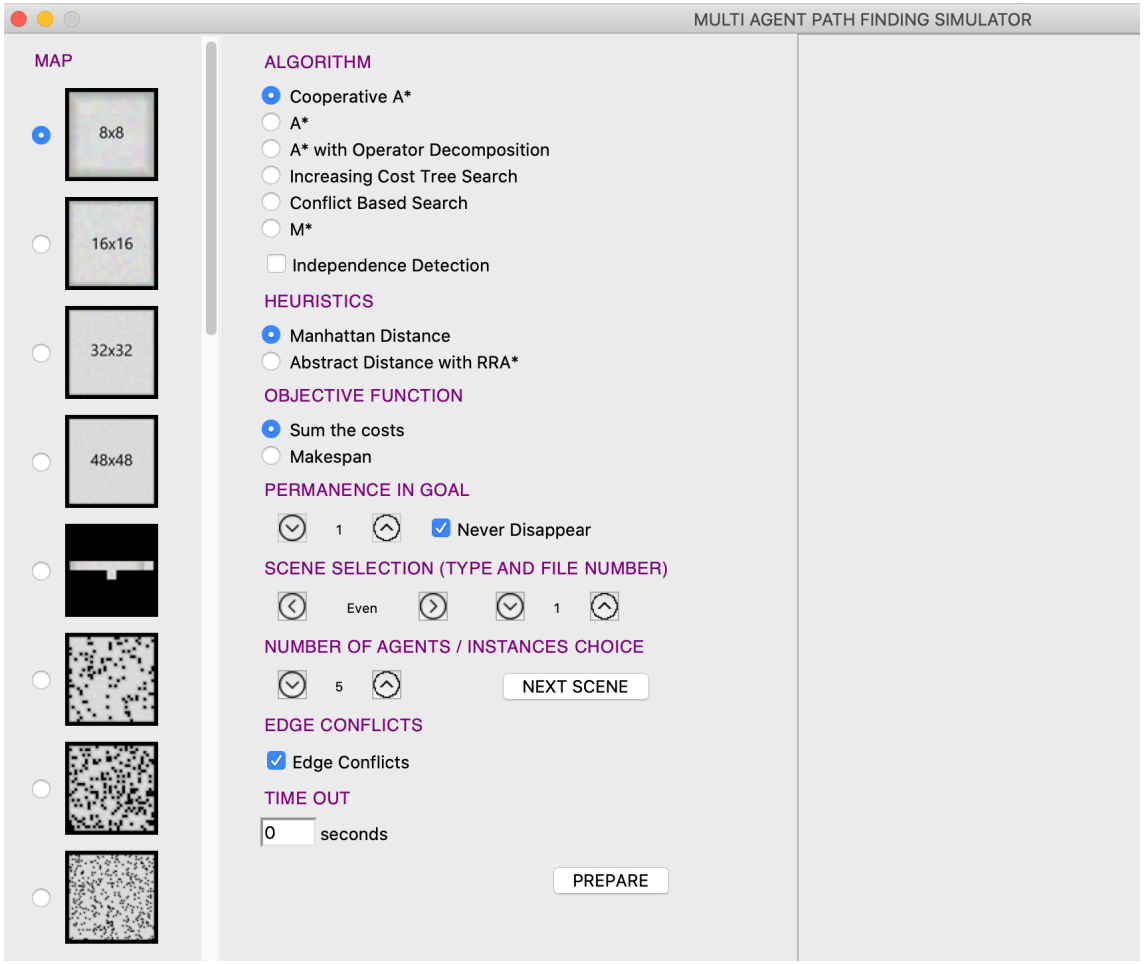
No established framework, but several different independent tools

- http://mapf.info/index.php/Main/Software
- https://github.com/PathPlanning/AA-SIPP-m
- …

Optional: software tools (2)

MAPF demo
(from a former MSc students)
https://github.com/MatteoAntoniazzi/MAPF

1. Installation
   Just clone the repository and run launcher.py
2. Definition of problems
   Graphical interface
3. Solution of problems
   Implementation of MAPF algorithms
4. Measurement of performance
   Metrics provided by the framework: time, generated nodes, …



Optional: homework

1. Install the MAPF demo tool
2. Run some random examples and visually investigate the behaviors of the algorithms we have seen in class
3. Do the following:
   - Change the code in order to be able to run all the algorithms on the same map with given start and destination locations for a given number of agents
   - Create 5 settings with the same 16x16 map with some obstacles and 5, 10, 15, 20, 25 agents
   - Compare the successes in finding a solution and the number of generated nodes of at least 3 algorithms in the 5 settings, considering a timeout of 60 seconds: draw graphs with number of agents on x axis and successes or number of generated nodes on the y axis
   - Which are the algorithms that scale better with respect to the number of agents?