ICAPS Summer School 2018: Introduction to Planning under Uncertainty in MDPs

Scott Sanner

UNIVERSITY OF TORONTO

Lecture Goals

1) To understand the ingredients of formal models for a range of applications in decision-making under uncertainty

2) To understand fundamental solution algorithms for these models and their properties

3) To understand how to build complex models (brief RDDL overview, more in lab)

4) Later MDP lectures: MCTS, RL and beyond

Planning under Uncertainty

- Definition:

  Computing sequences of actions to obtain occasional rewards in a known, stochastic environment

Reinforcement Learning (RL)

- Definition:

  Learning to act from periodic rewards in an unknown, stochastic environment

Applications

Elevator Control

- **Concurrent Actions**
  - Elevator: up/down/stay
  - 6 elevators: 3^6 actions
- **Dynamics**:
  - Random arrivals (e.g., Poisson)
- **Objective:**
  - Minimize total wait
  - (Requires being proactive about future arrivals)
- **Constraints:**
  - People might get annoyed if elevator reverses direction

Two-player Games

- **Othello / Reversi**
  - Solved by Logistello!
  - Monte Carlo RL (self-play) + Logistic regression + Search

- **Backgammon**
  - Solved by TD-Gammon!
  - Temporal Difference (self-play) + Artificial Neural Net + Search

- **Go**
  - Learning + Search
  - AlphaGo (MCTS + deep learning) recently the world champion

Multi-player Games: Poker

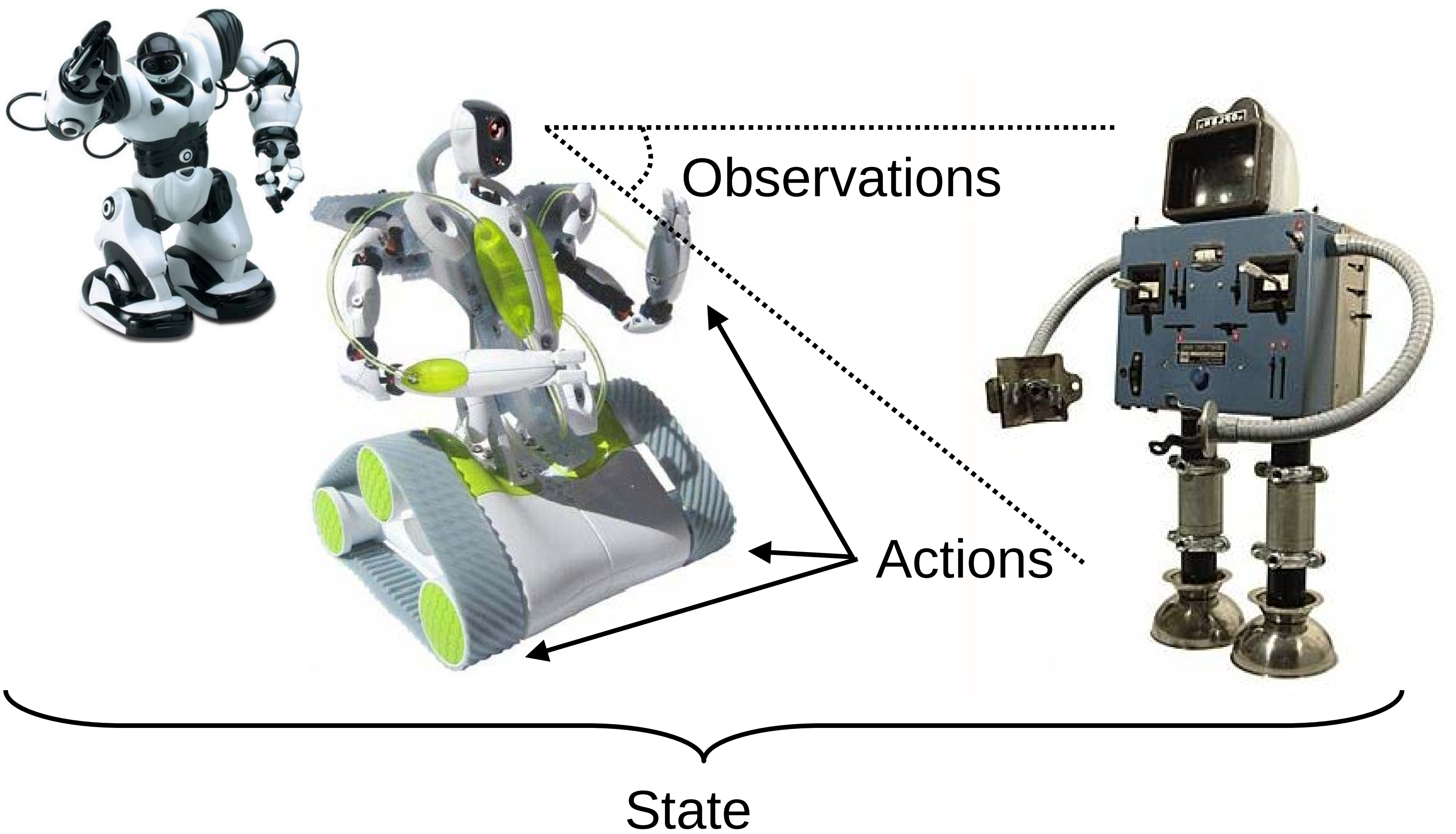
- **Multiagent (adversarial)**
  - Opponent may abruptly change strategy
  - Might prefer best outcome for *any* opponent strategy (e.g, a Nash equilibrium)
- **Multiple rounds (sequential)**
- **Partially observable!**
  - Earlier actions may reveal information
  - Or they may not (bluff)

DARPA Grand Challenge

- **Autonomous mobile robotics**
  – Extremely complex task, requires expertise in vision, sensors, real-time operating systems
- **Partially observable**
  – e.g., only get noisy sensor readings
- **Model unknown**
  – e.g., steering response in different terrain


How to model these problems?

Observations, States, & Actions



Observations

- **Observation set O**
  - Perceptions, e.g.,
    - Distance from car to edge of road
    - My opponent’s bet in Poker

States

- **State set S**
  - At any point in time, system is in some state
    - Actual distance to edge of road
    - My opponent's hand of cards in Poker

Agent Actions

- **Action set A**
  - Actions could be *concurrent*
  - If k actions, $\mathbf{A} = \mathbf{A_1} \times \ldots \times \mathbf{A_k}$
    - Schedule all deliveries to be made at 10am

Agent Actions

- **Action set A**
  - All actions need not be under agent control
    - Other agents, e.g.,
      - Alternating turns: Poker, Othello
      - Concurrent turns: Highway Driving, Soccer
    - *Exogenous events* due to *Nature*, e.g.,
      - Random arrival of person waiting for elevator
      - Random failure of equipment
      - If uncontrolled, model as random variables

Observation Function

- How to relate states and observations?
  - ***Not observable*:**
    - **O =** $\emptyset$
    - e.g., heaven vs. hell
      - » only get feedback once you meet St. Pete
  - ***Fully observable*:**
    - **S** $\leftrightarrow$ **O** … the case we focus on!
    - e.g., many board games,
      - » Othello, Backgammon, Go
  - ***Partially observable*:**
    - all remaining cases
    - e.g., driving a car, Poker, the real world!

Recap

- So far
  - Actions
  - States
  - Observations

- How to map between
  - Previous states, actions, and future states?
  - States and observations?
  - States, actions and rewards?
  - Sequences of rewards and optimization criteria?

Transition Function

- How do actions take us between states?
  - T(s,a,s') encodes P(s'|s,a)
  - Some properties
    - *Stationary*: T does not change over time
    - *Markovian*: Only depends on previous state / action
    - If T not Markovian or stationary
      - can sometimes achieve by augmenting state description
        - » e.g., elevator traffic differs throughout day… encode time in state to make T Markovian!

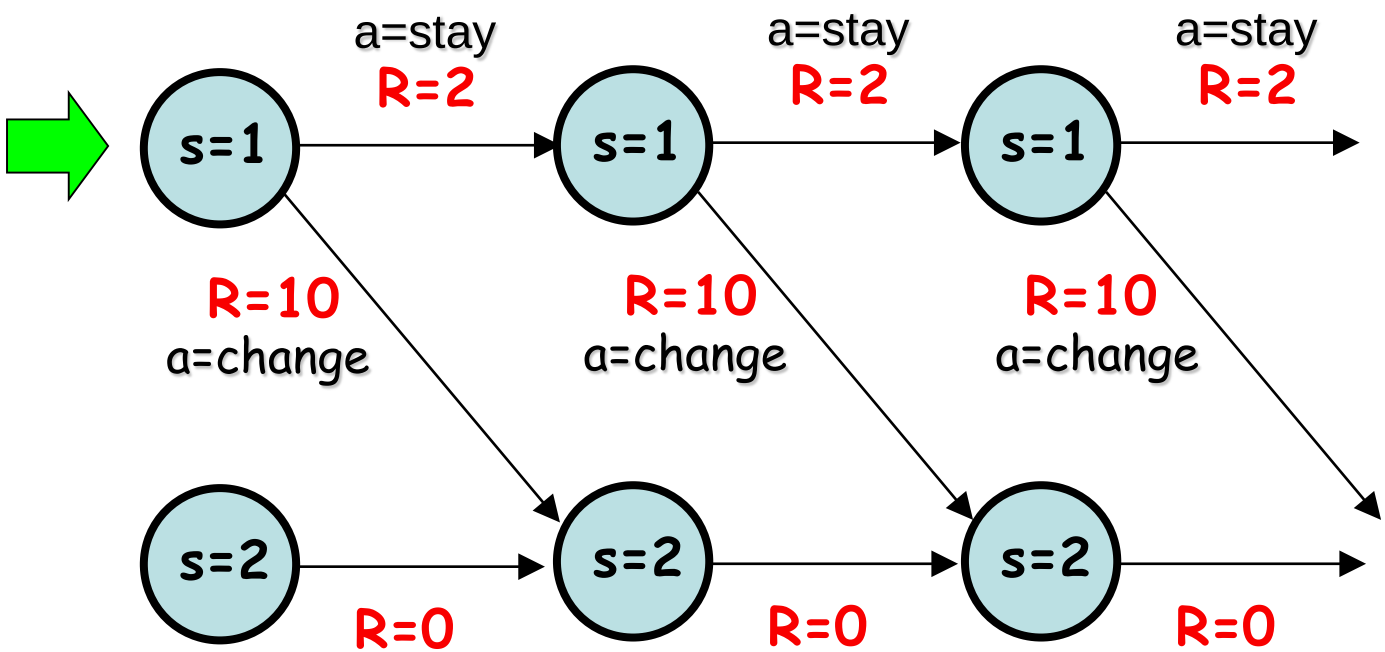
Goals and Rewards

- Goal-oriented rewards
  - Assign any reward value s.t. *R(success) > R(fail)*
  - Can have negative costs *C(a)* for action *a*

- What if multiple (or no) goals?
  - How to specify preferences?
  - *R(s,a)* assigns utilities to each state *s* and action *a*
    - Then *maximize expected reward (utility)*

But, how to trade off rewards over time?

Optimization: Best Action when s=1?



- Must define objective criterion to optimize!
  - How to trade off immediate vs. future reward?
  - E.g., use discount factor γ (try γ=.9 vs. γ=.1)

Trading Off Sequential Rewards

- Sequential-decision making objective
  - Horizon
    - *Finite*: Only care about h-steps into future
    - Infinite: Literally; will act same today as tomorrow
  - How to trade off reward over time?
    - *Expected average cumulative return*
    - *Expected discounted cumulative return*
      - Use discount factor γ
      - Reward t time steps in future discounted by γ^t

Recap

- Model so far
  - Actions A
  - States S
  - Observation O
  - Transition function T: P(s'|s,a)
  - Observation function Z: P(o'|s,a) – *POMDPs only*
  - Reward function: R(s,a)
  - Optimization criteria
- But are the above
  - Known or unknown?

Knowledge of Environment

- ***Model-known*:**
  - Know observation, transition, & reward functions
  - Called: *Planning (under uncertainty)*
    - Planning generally assumed to be goal-oriented
    - *Decision-theoretic* if maximizing expected utility

- ***Model-free*:**
  - ≥1 unknown: observation, transition, & reward functions
  - Called: *Reinforcement learning*
    - Have to interact with environment to obtain samples

- ***Model-based*: approximate model in model-free case**
  - Permits hybrid planning and learning

Saves expensive interaction!

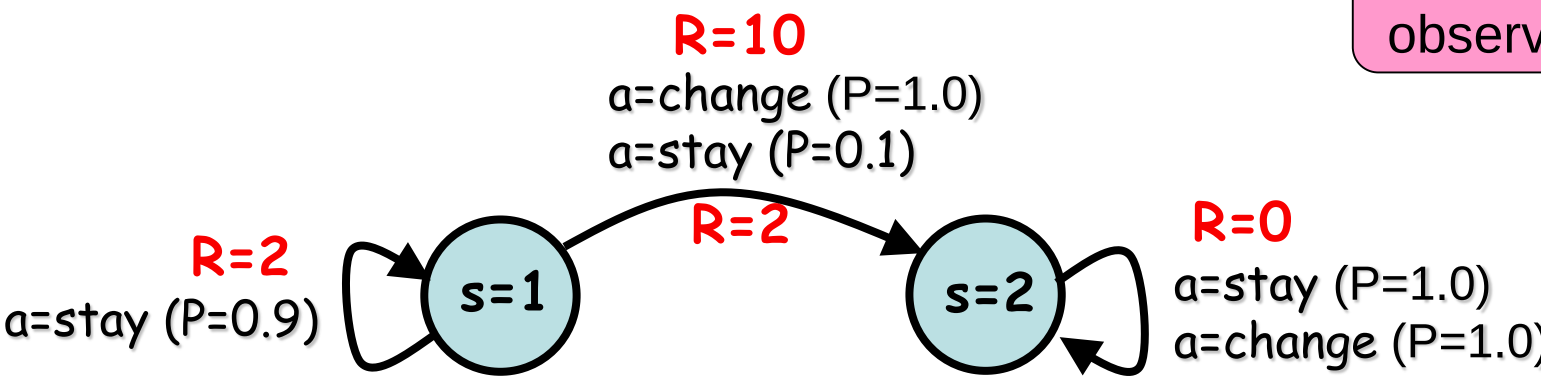
Finally a Formal Model

- **Define the previous model**
  - MDP: ⟨ **S, A, T, R** ⟩
  - POMDP: ⟨ **S, A, O, Z, T, R** ⟩
  - Whether known / unknown
- **Characterize the solutions**
  - And efficiently find them!

Model-based Solutions to MDPs

MDPs ⟨S,A,T,R⟩

Note: fully observable

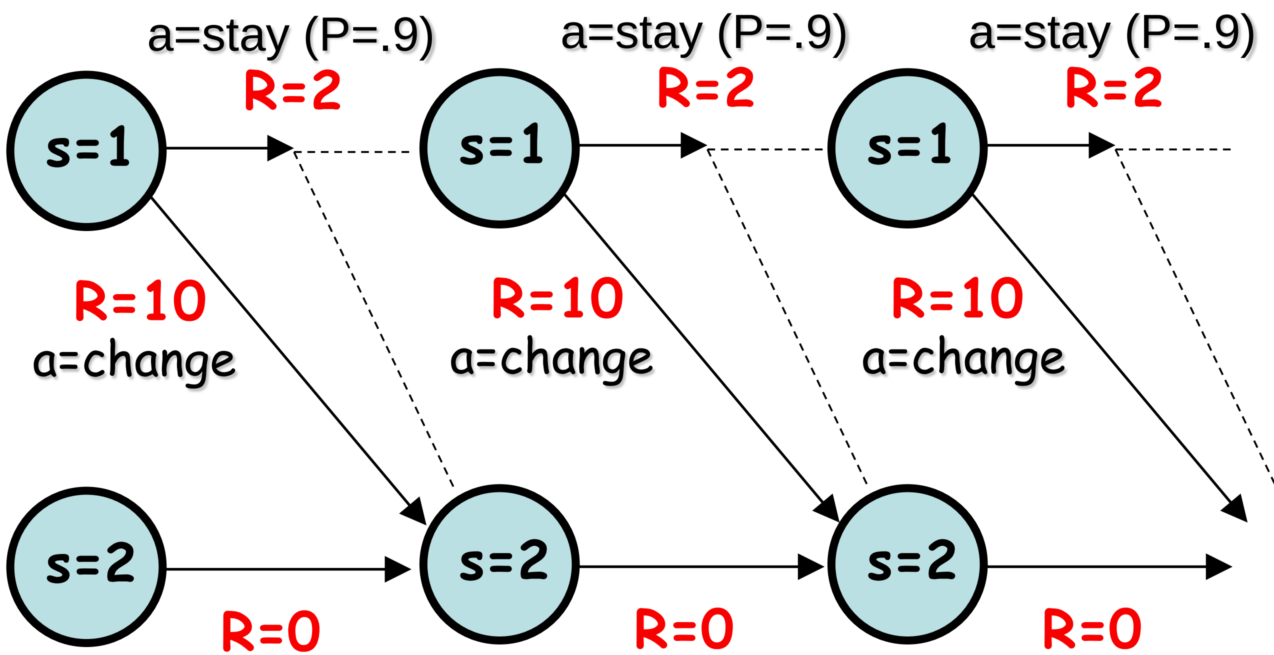


- **S** = {1,2}; **A** = {stay, change}
- **R**eward
  - R(s=1,a=stay) = 2
  - …
- **T**ransitions
  - T(s=1,a=stay,s'=1) = P(s'=1 | s=1, a=stay) = .9
  - …

How to act in an MDP?

Define policy
π: S → A

What's the best Policy?



- Must define reward criterion to optimize!
  - Discount factor γ important (γ=1.0 vs. γ=0.1)

MDP Policy, Value, & Solution

- Define *value of a policy* π:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \middle| s = s_0\right]$$

- Tells how much value you expect to get by following π starting from state *s*

- Allows us to define optimal solution:
  - Find optimal policy π^* that maximizes value
  - Surprisingly: $\exists \pi^*. \forall s, \pi.\ V_{\pi^*}(s) \geq V_\pi(s)$
  - Furthermore: always a *deterministic* π^*

Value Function → Policy

- Given arbitrary value V (optimal or not)…
  - A *greedy policy* π_V takes action in each state that maximizes expected value w.r.t. *V*:

$$\pi_V(s) = \arg\max_a \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \right\}$$

  - If can act so as to obtain *V* after doing action *a* in state *s*, π_V guarantees *V(s)* in expectation

If *V* not optimal, but a *lower bound* on *V**, π_V guarantees at least that much value!

Value Iteration: from finite to ∞ decisions

- Given optimal (*t-1*)*-stage-to-go* value function
- How to act optimally with *t* decisions?
  - Take action *a* then act so as to achieve V^{t-1} thereafter

$$Q^t(s,a) := R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')$$

  - What is expected value of best action *a* at decision stage *t*?

$$V^t(s) := \max_{a \in A} \left\{ Q^t(s,a) \right\}$$

  - At ∞ horizon, converges to V^*

$$\lim_{t \to \infty} \max_s \left| V^t(s) - V^{t-1}(s) \right| = 0$$

  - This *value iteration* solution know as *dynamic programming (DP)*

Make sure you can derive these equations from first principles!

Bellman Fixed Point

- Define *Bellman backup* operator *B*:

$$\overbrace{(B\,\underbrace{V}_{V^{t-1}})}^{V^{t}}(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right\}$$

- $\exists$ an optimal value function V* and an optimal deterministic greedy policy $\pi^* = \pi_{V^*}$ satisfying:

$$\forall s.\ V^*(s) = (B\,V^*)(s)$$

Bellman Error and Properties

- Define *Bellman error BE*:

$$(BE\,V) = \max_s |(B\,V)(s) - V(s)|$$

- Clearly:

$$(BE\,V^*) = 0$$

- Can prove *B* is a contraction operator for *BE*:

$$(BE\,(B\,V)) \leq \gamma (BE\,V)$$

Hmmm…. Does this suggest a solution?

Value Iteration: in search of fixed-point

- Start with arbitrary value function V^0
- Iteratively apply Bellman backup

$$V^t(s) = (B\,V^{t-1})(s)$$

Look familiar? Same DP solution as before.

- Bellman error decreases on each iteration
  - Terminate when

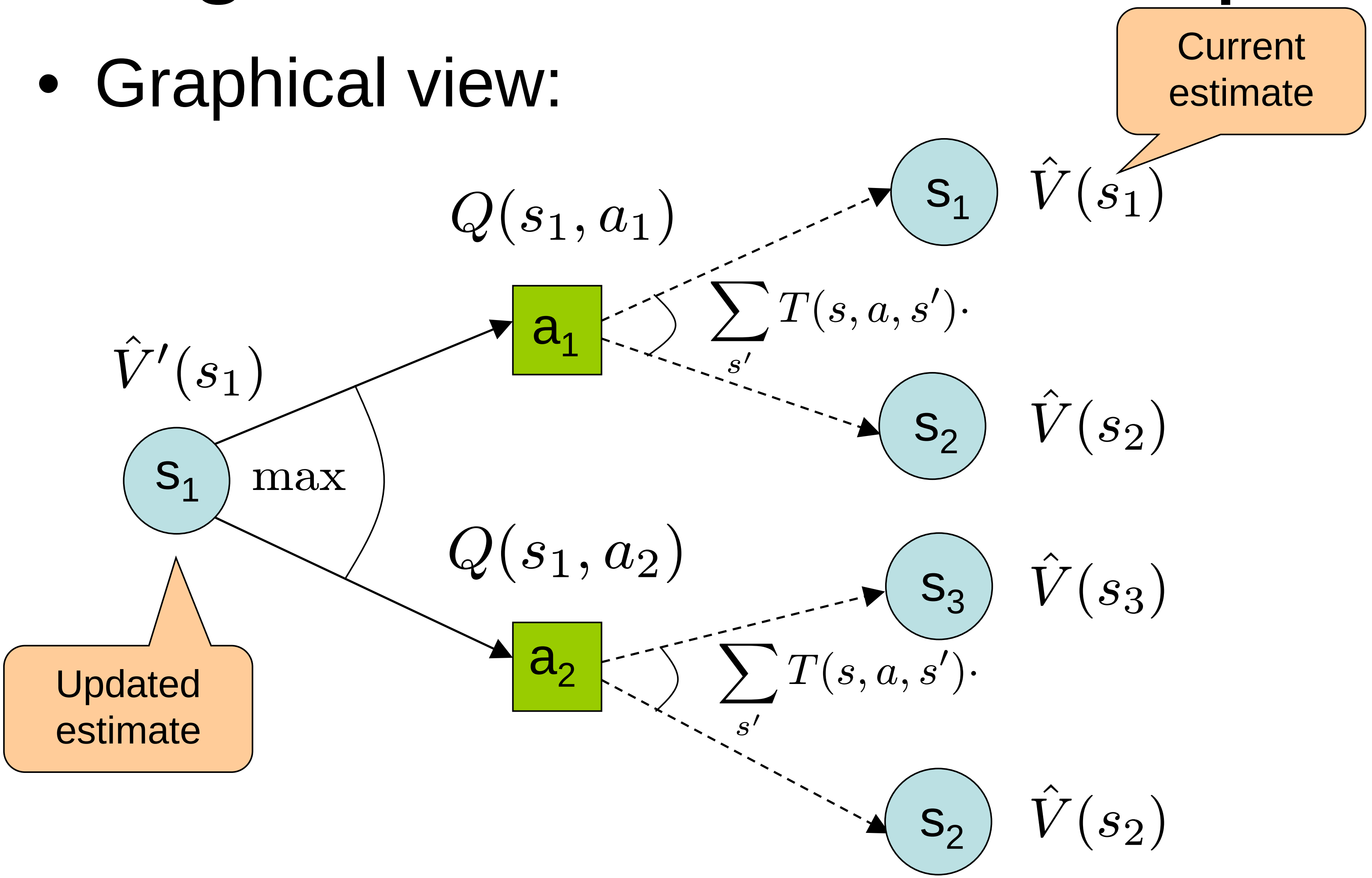
$$\max_s |V^t(s) - V^{t-1}(s)| < \frac{\epsilon(1-\gamma)}{2\gamma}$$

  - Guarantees ε-optimal value function
    - i.e., V^t within ε of V^* for all states

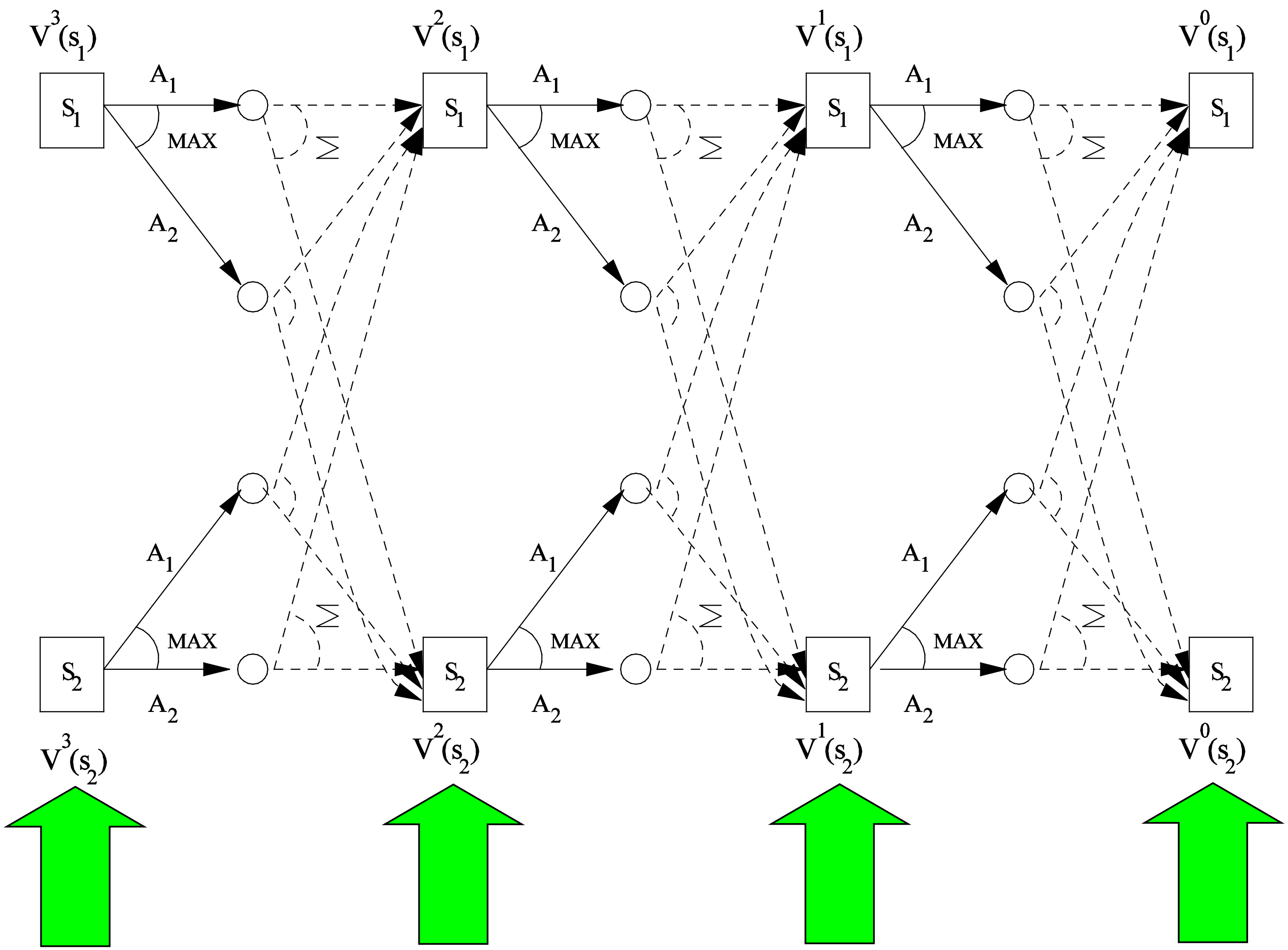
Precompute maximum number of steps for ε?

Single DP Bellman Backup

- Graphical view:

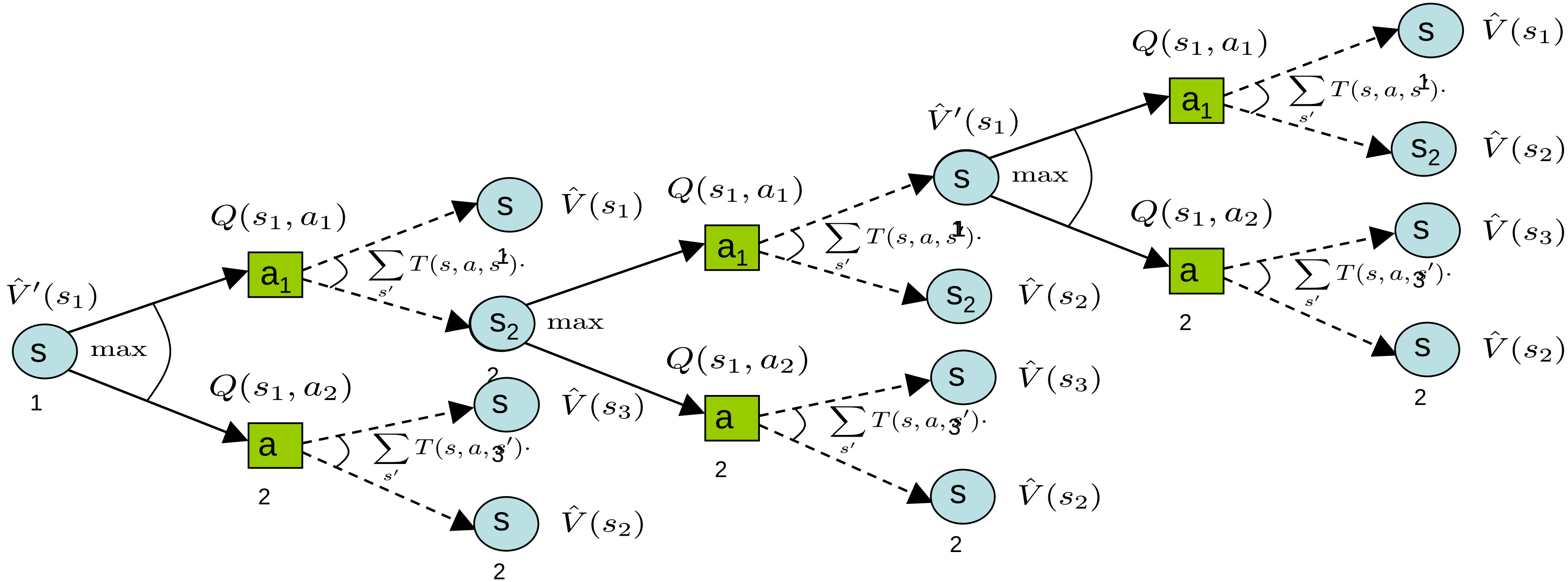


Synchronous DP Updates (VI)



Asynchronous DP Updates

- Or… can update states in any order:



- Still provably converges!

Question:
how to order updates to converge quickly?

Real-time Dynamic Programming

- ***Reachability*** and drawbacks of synch. DP (VI)

  - Better to think of ***relevance*** to optimal policy

- RTDP focuses async. updates on relevant states!

Policy Evaluation

- Given π, how to derive V_π?
- *Matrix inversion*
  - Set up linear equality (no max!) for each state

$$\forall s.\ V_\pi(s) = \left\{ R(s,\pi(s)) + \gamma \sum_{s'} T(s,\pi(s),s') V_\pi(s') \right\}$$

  - Can solve linear system in vector form as follows

$$V_\pi = R_\pi (I - \gamma T_\pi)^{-1}$$

Guaranteed invertible.

- *Successive approximation*
  - Essentially value iteration with fixed policy
  - Initialize V_π^0 arbitrarily

$$V_\pi^t(s) := \left\{ R(s,\pi(s)) + \gamma \sum_{s'} T(s,\pi(s),s') V_\pi^{t-1}(s') \right\}$$

  - Guaranteed to converge to V_π

Policy Iteration

1. *Initialization:* Pick an arbitrary initial decision policy $\pi_0 \in \Pi$ and set $i = 0$.

2. *Policy Evaluation:* Solve for V_{π_i} (previous slide).

3. *Policy Improvement:* Find a new policy π_{i+1} that is a greedy policy w.r.t. V_{π_i}

   (i.e., $\pi_{i+1} \in \arg\max_{\pi \in \Pi} \{R_\pi + \gamma T_\pi V_{\pi_i}\}$ with ties resolved via a total precedence order over actions).

4. *Termination Check:* If $\pi_{i+1} \neq \pi_i$ then increment i and go to step 2 else return π_{i+1}.

Between Value and Policy Iteration

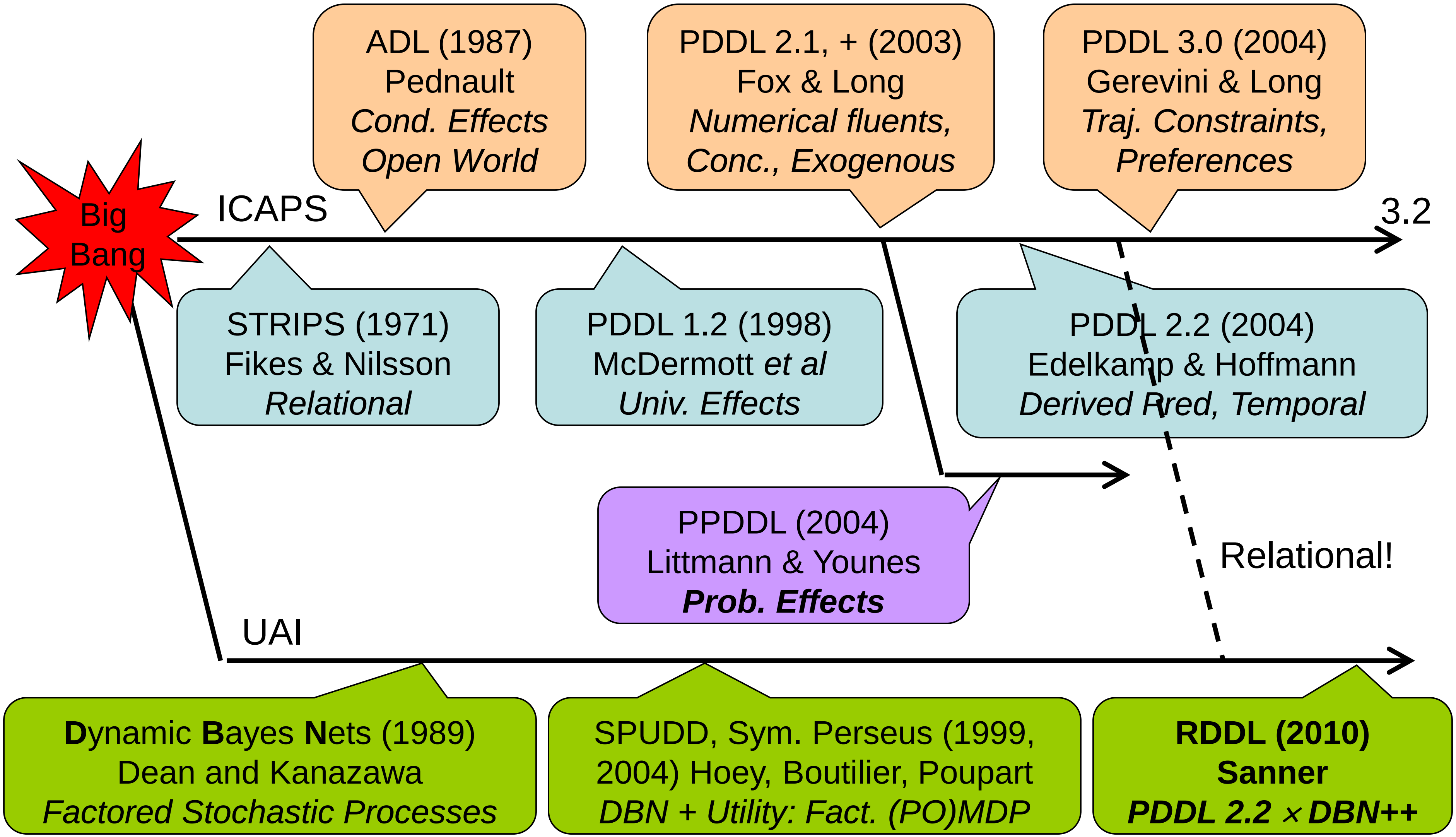
- *Value iteration*
  - Each iteration seen as doing 1-step of policy evaluation for current greedy policy
  - Bootstrap with value estimate of previous policy

- *Policy iteration*
  - Each iteration is full evaluation of V_π for current policy π
  - Then do greedy policy update

- *Modified policy iteration*
  - Like policy iteration, but $V_{\pi i}$ need only be closer to V* than $V_{\pi i-1}$
    - Fixed number of steps of successive approximation for $V_{\pi i}$ suffices when bootstrapped with $V_{\pi i-1}$
  - Typically faster than VI & PI in practice

Advanced (PO)MDP Modeling with RDDL

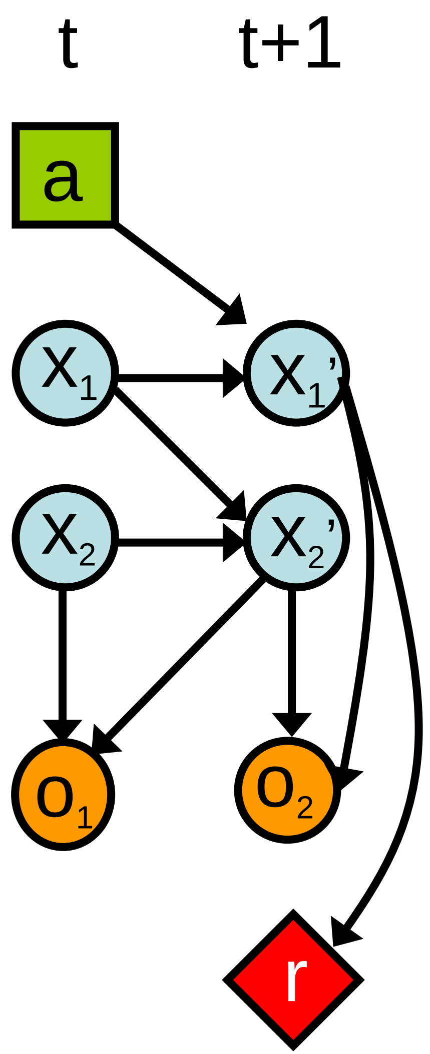
A Brief History of (ICAPS) Time



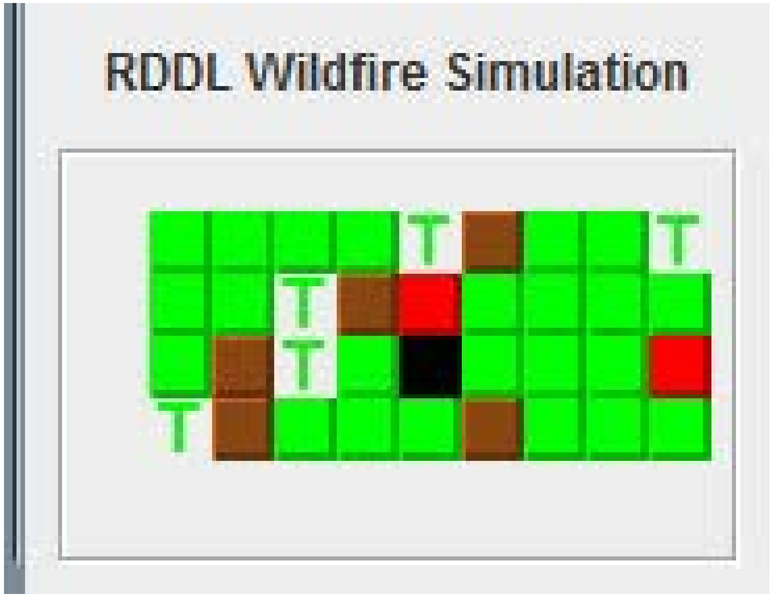
PDDL history from: http://ipc.informatik.uni-freiburg.de/PddlResources

What is RDDL?

- Relational Dynamic Influence Diagram Language
  - Relational [DBN + Influence Diagram]
  - Everything is a fluent!
    - states
    - observations
    - actions
  - Conditional distributions are *probabilistic programs*



Wildfire Domain (today's lab)



- Contributed by Zhenyu Yu (School of Economics and Management, Tongji University)
  - Karafyllidis, I., & Thanailakis, A. (1997). *A model for predicting forest fire spreading using gridular automata*. Ecological Modelling, 99(1), 87-97.

Wildfire in RDDL

```
cpfs {

      burning'(?x, ?y) =
            if ( put-out(?x, ?y) )
                  then false
            else if (~out-of-fuel(?x, ?y) ^ ~burning(?x, ?y))
                  then Bernoulli( 1.0 / (1.0 + exp[4.5 - (sum_{?x2: x_pos, ?y2: y_pos}
                                               (NEIGHBOR(?x, ?y, ?x2, ?y2) ^ burning(?x2, ?y2)))]) )
            else
                  burning(?x, ?y); // State persists

      out-of-fuel'(?x, ?y) = out-of-fuel(?x, ?y) | burning(?x,?y);

};

reward =
     [sum_{?x: x_pos, ?y: y_pos} [ COST_CUTOUT*cut-out(?x, ?y) ]]
   + [sum_{?x: x_pos, ?y: y_pos} [ COST_PUTOUT*put-out(?x, ?y) ]]
   + [sum_{?x: x_pos, ?y: y_pos} [ COST_NONTARGET_BURN*[ burning(?x, ?y) ^ ~TARGET(?x, ?y) ]]]
   + [sum_{?x: x_pos, ?y: y_pos}
          [ COST_TARGET_BURN*[ (burning(?x, ?y) | out-of-fuel(?x, ?y)) ^ TARGET(?x, ?y) ]]];
```

Facilitating Model Development by Writing Simulators: Relational Dynamic Influence Diagram Language (RDDL)

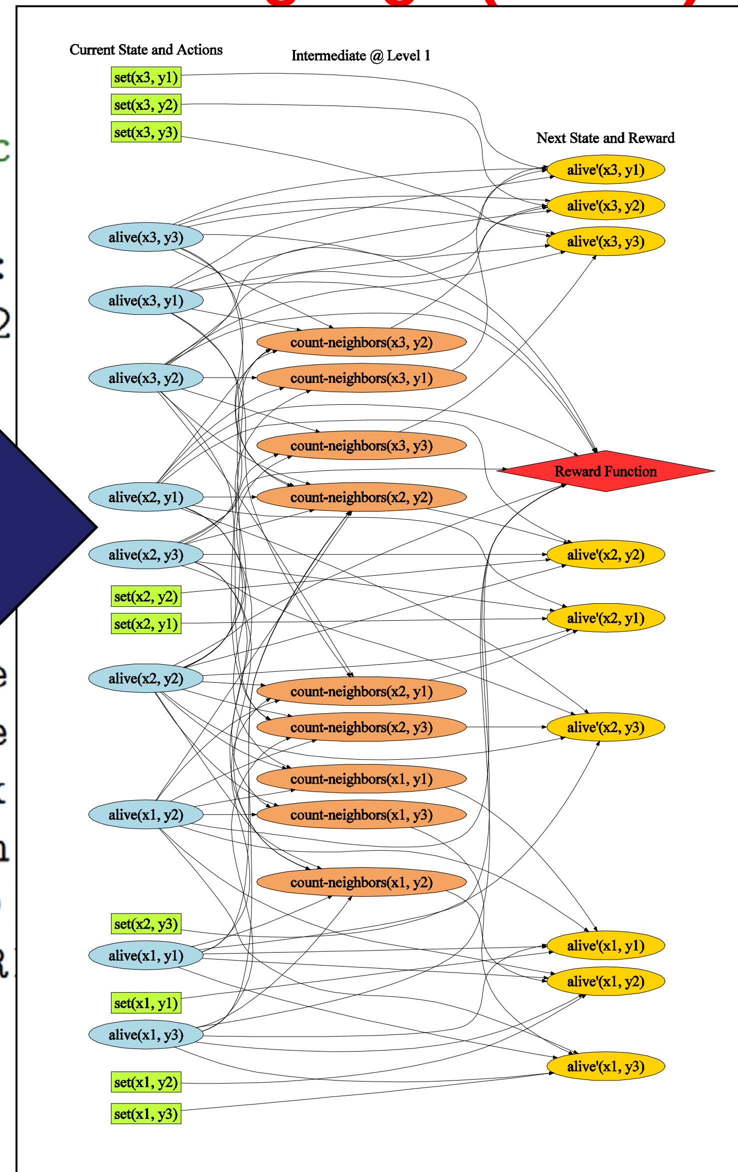
```
// Store alive-neighbor count for eac
count-neighbors(?x,?y) =
    KronDelta(sum_{?x2 : x_pos, ?y2 :
              [NEIGHBOR(?x,?y,?x2,?y2

// Determine whether
alive'(?x,?y) = if (

            else
                      ^ (coun    ne
                      ^ (count-ne
                     | [~alive(?x
                       ^ (count-n
                     | set(?x,?y)
            hen Bernoulli(PROB_R
            else Bernoulli(1.0 -
```

Automatic Translation

Write **probabilistic programs** for transitions



RDDLSim Software

Open source & online at
http://code.google.com/p/rddlsim/

RDDL Software Overview

- BNF grammar and parser
- Simulator
- Automatic compilation / translations
  - LISP-like format (easier to parse)
  - SPUDD & Symbolic Perseus (boolean subset)
  - Ground PPDDL (boolean subset)
- Client / Server
  - Java and C/C++ sample clients
  - Evaluation scripts for log files
- **Visualization**
  - DBN Visualization
  - Domain Visualization – *see* how your planner is doing

Initial Use of RDDL

- Have run two major competitions at ICAPS

- Translations to draw in different communities
  - UAI Factored MDP / POMDP community
  - ICAPS PPDDL community
  - 11 competitors in 2011, 6 competitors in 2014

- Competitions drive research progress!
  - Historically, ICAPS focused on deterministic replanning
  - With RDDL + new domains, **MCTS dominates**
    (namely PROST system by Thomas Keller *et al*)

Recap: Lecture Goals

1) To understand the ingredients of formal models for a range of applications in decision-making under uncertainty

2) To understand fundamental solution algorithms for these models and their properties

3) To understand how to build complex models (brief RDDL overview, more in lab)

4) Upcoming MDP lectures: MCTS, RL, …