Reinforcement Learning

Introduction & Model-Based Learning

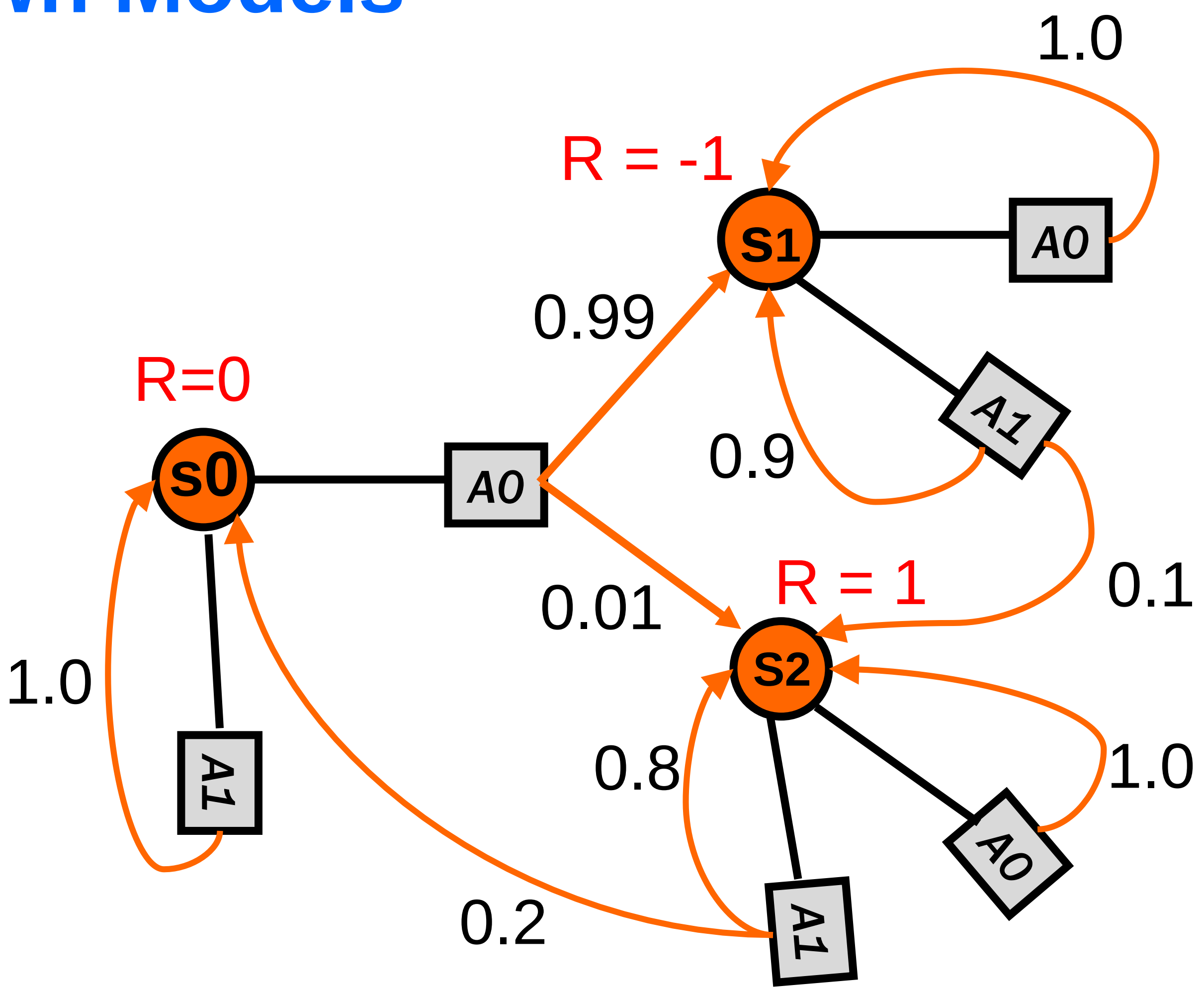
Alan Fern

Reinforcement Learning

There are at least two situations where reinforcement learning is useful.

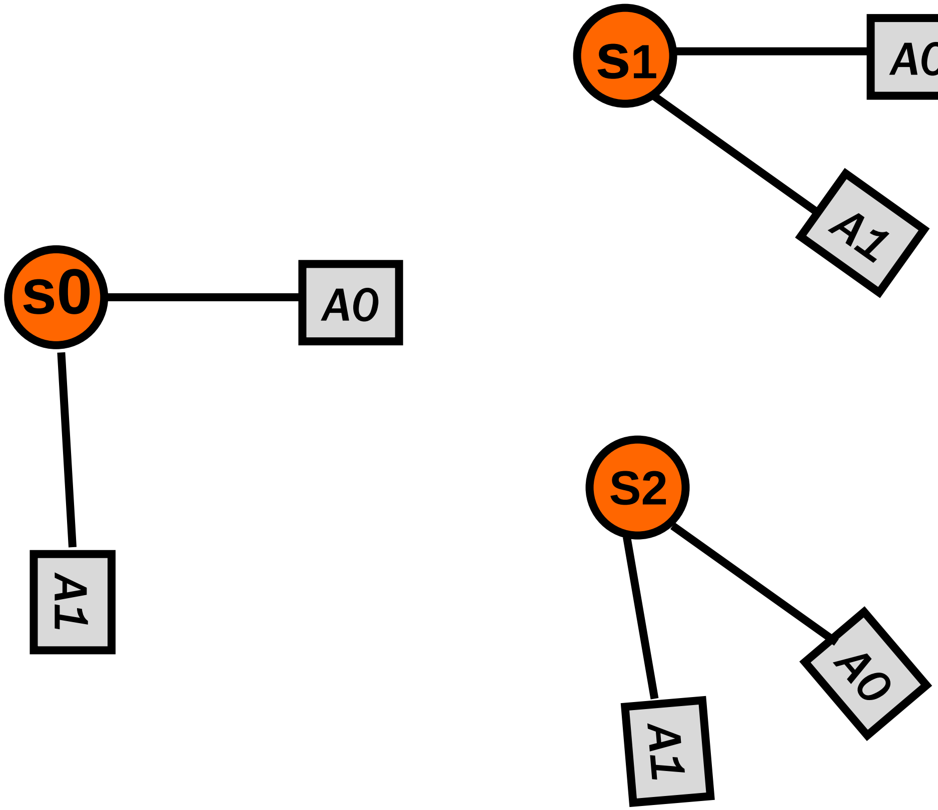
1. When the environment model (MDP) is unknown.
2. When the environment is enormous.

Known Models



Given a moderately-sized MDP model, we can use value iteration or value iteration to solve it.

Unknown Models



What if we don't know the reward and transition functions?
(Like in many real-world domains.)

But we can take actions and observe their effects.

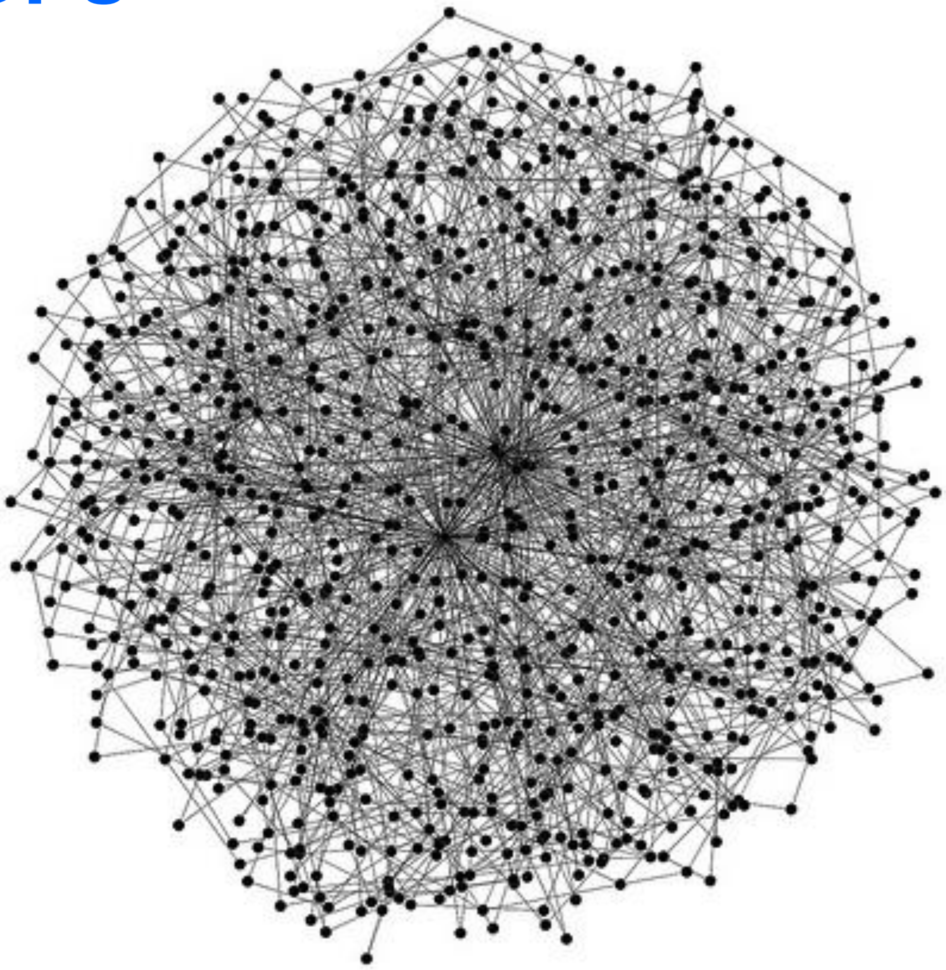
Unknown MDP Model

- In many real-world domains it is difficult to hand-code an MDP model that is sufficiently accurate.

- **Option 1:** Hand-code a parameterized MDP model and then manually collect data to tune the model parameters.
  - E.g. certain probabilities may be unknown, but could be inferred from appropriately collected data

- **Option 2:** Reinforcement learning can do this automatically, or learn a policy directly without explicit model learning.

Enormous MDPs



What if an MDP is enormous, regardless of whether we know the model or not?

Enormous Worlds

- We have considered basic model-based planning algorithms
- **<u>Model-based planning</u>**: assumes MDP model is available
  - Methods we learned so far are at least poly-time in the number of states and actions
  - Difficult to apply to large state and action spaces (though this is a rich research area)
- We will consider various methods for overcoming this issue

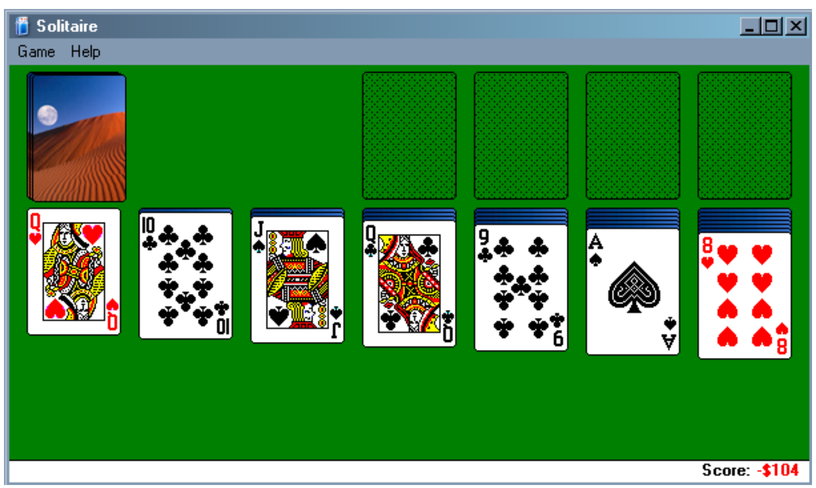
Approaches for Enormous Worlds

- **Planning with compact MDP representations**
  1. Define a language for compactly describing an MDP
     - MDP is exponentially larger than description
     - E.g. via Dynamic Bayesian Networks
  2. Design a planning algorithm that directly works with that language
- Scalability is still an issue
- Can be difficult to encode the problem you care about in a given language
- May study in last part of course

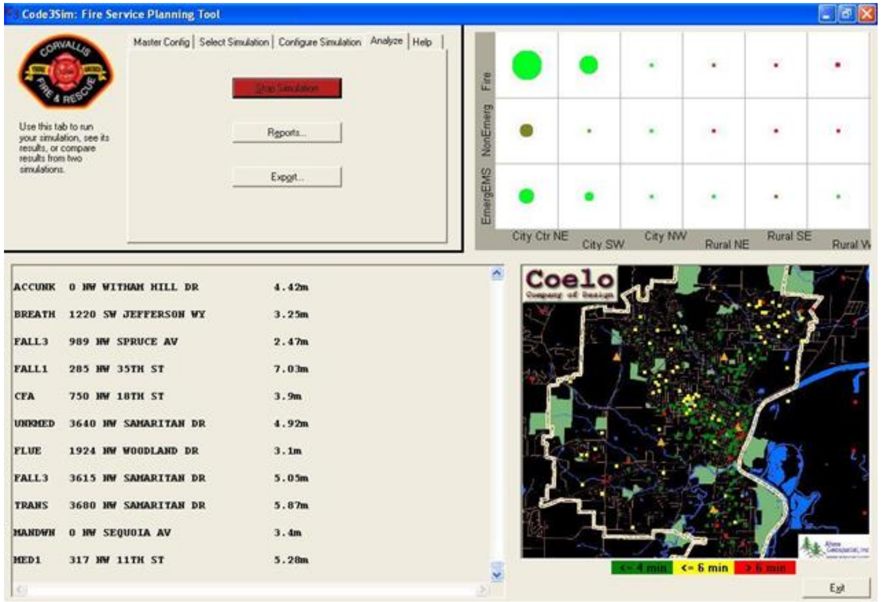
Approaches for Enormous Worlds: Monte-Carlo Planning

- Often a simulator of a planning domain is available or can be learned/estimated from data
  - Will study later in the course

Klondike Solitaire



Fire & Emergency Response



Approaches for Large Worlds

- **<u>Reinforcement learning w/ function approx.</u>**
  1. Have a learning agent directly interact with environment
  2. Learn a compact description of policy or value function

- Often works quite well for large problems
  - Robotics
  - Networking
  - Games (e.g. Atari)
  - ….

Reinforcement Learning

- No knowledge of environment
  - Can only act in the world and observe states and reward
- Many factors make RL difficult:
  - Actions have non-deterministic effects
    - Which are initially unknown
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment)
  - World is large and complex
- Imagine trying to learn to play solitaire or chess without being told the rules or objective

Model-Based vs. Model-Free RL

- *Model based approach to RL:*
  - learn the MDP model, or an approximation of it
  - use it to find an optimal policy
- *Model free approach to RL:*
  - directly learn a value function or policy without explicitly learning a model
  - useful when model is difficult to represent and/or learn, or when model is too large for our optimization algorithms
- We will consider both types of approaches
  - Will start with model-based
  - But will put more emphasis on model-free

Small vs. Huge MDPs

- We will first cover RL methods for small MDPs
  - MDPs where the number of states and actions is reasonably small
  - These algorithms will inspire more advanced methods

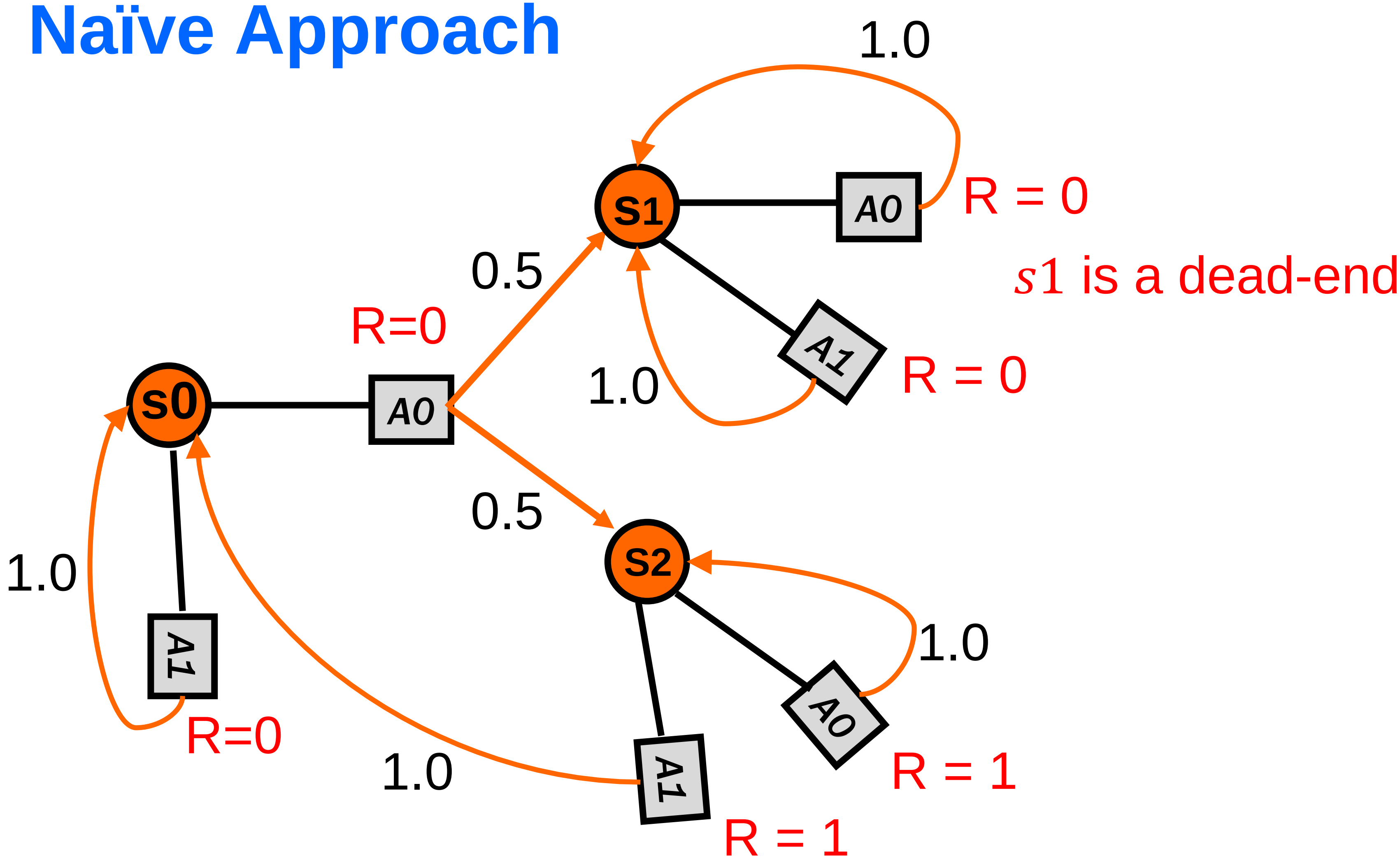
- Later we will cover algorithms for huge MDPs
  - Function Approximation Methods
  - Policy Gradient Methods

Naïve Model-Based Approach

1. Act Randomly for a (long) time
2. Learn
   - Transition function
   - Reward function
3. Apply value/policy iteration to get policy
4. Follow resulting policy thereafter.

Will this work?

Naïve Approach



Can’t learn after entering a dead-end.

RL theory generally assumes no dead-ends.

Naïve Model-Based Approach

1. Act Randomly for a (long) time
2. Learn
   - Transition function
   - Reward function
3. Apply value/policy iteration to get policy
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no “dead-ends”)

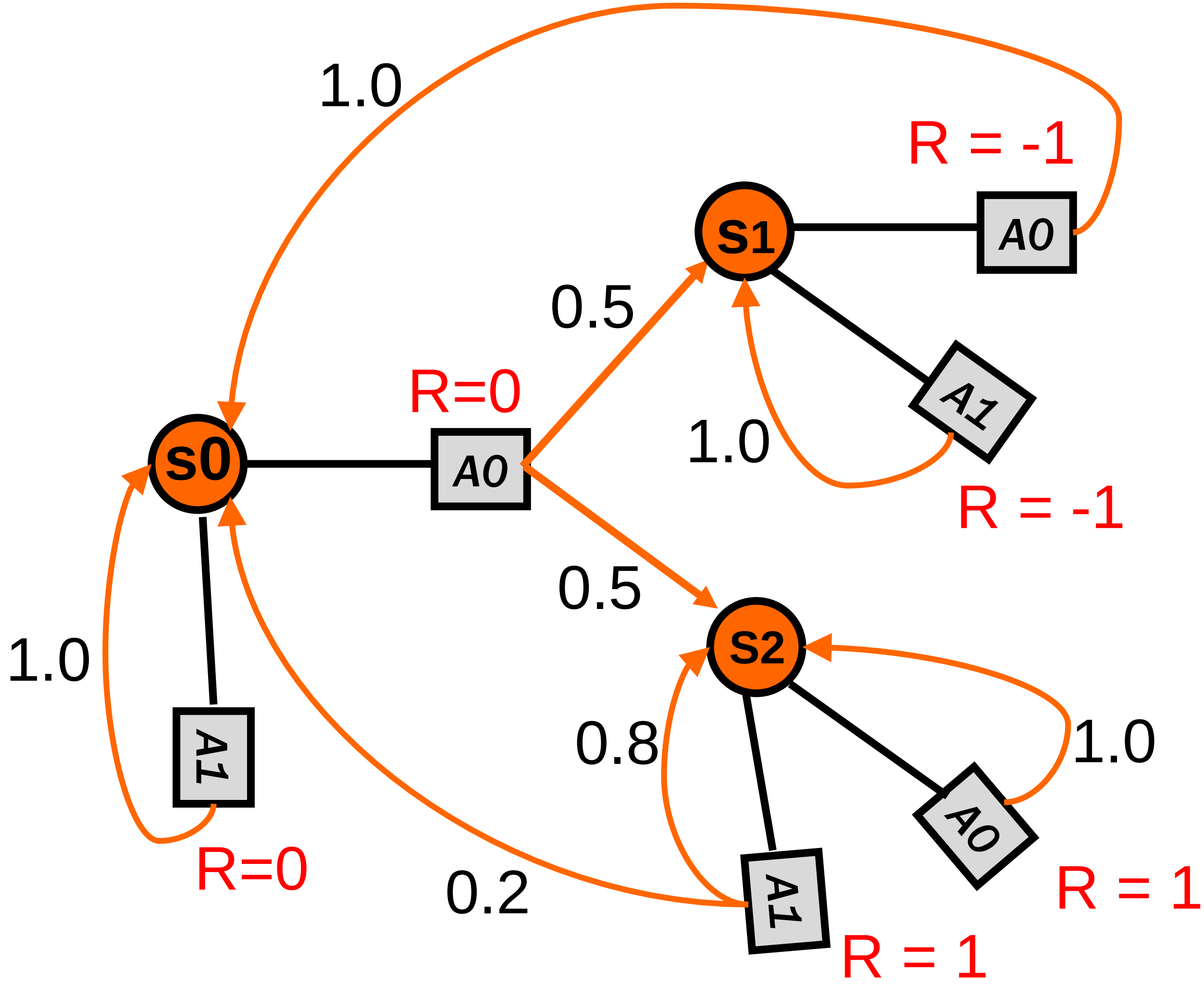
Any problems? We will act randomly for a long time before exploiting what we know.

Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

Will this work?

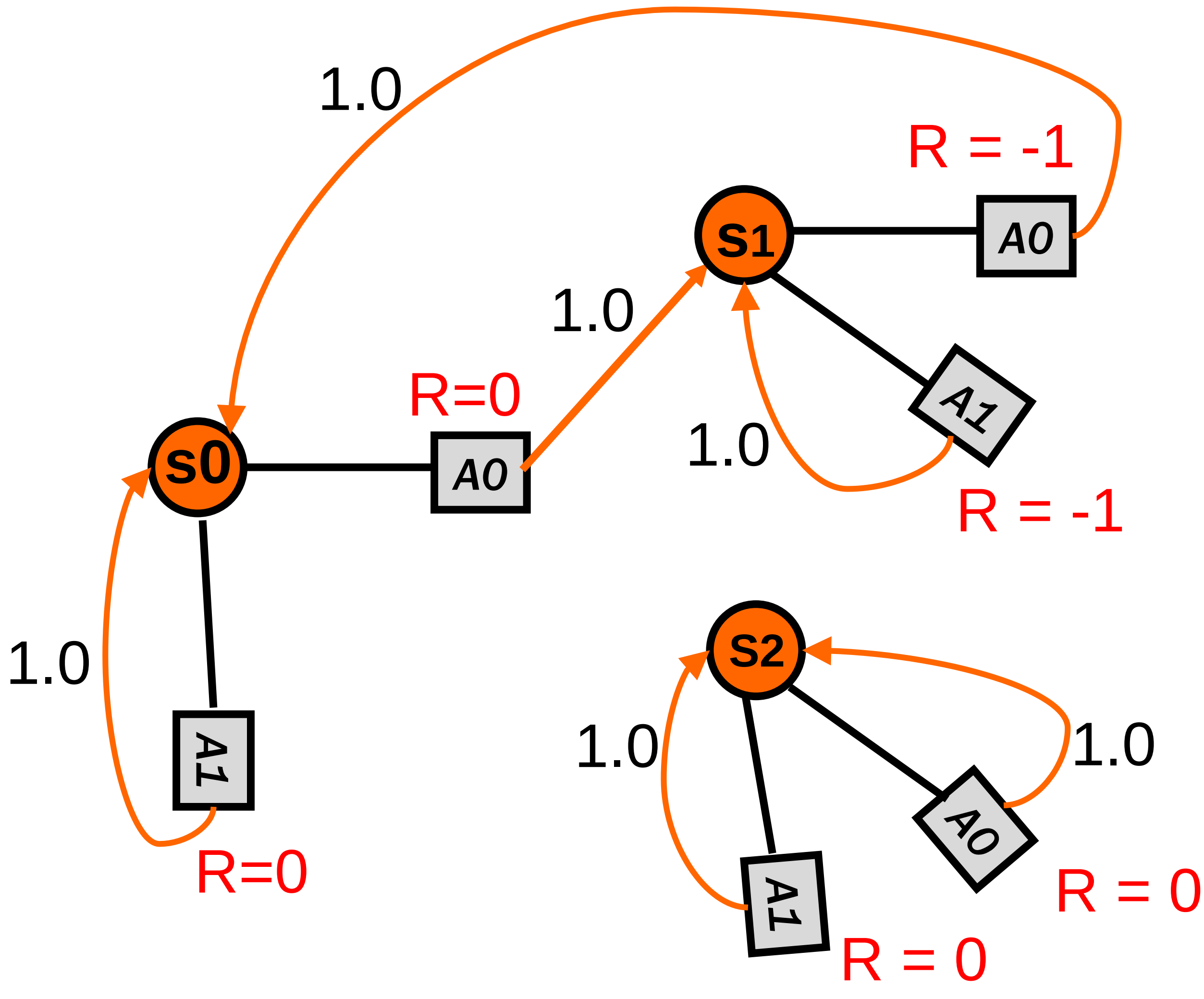
Revision of Naïve Approach



True MDP

Suppose our algorithm learns from s0,A1,s0,A0,s1,A0

Revision of Naïve Approach



Learned MDP

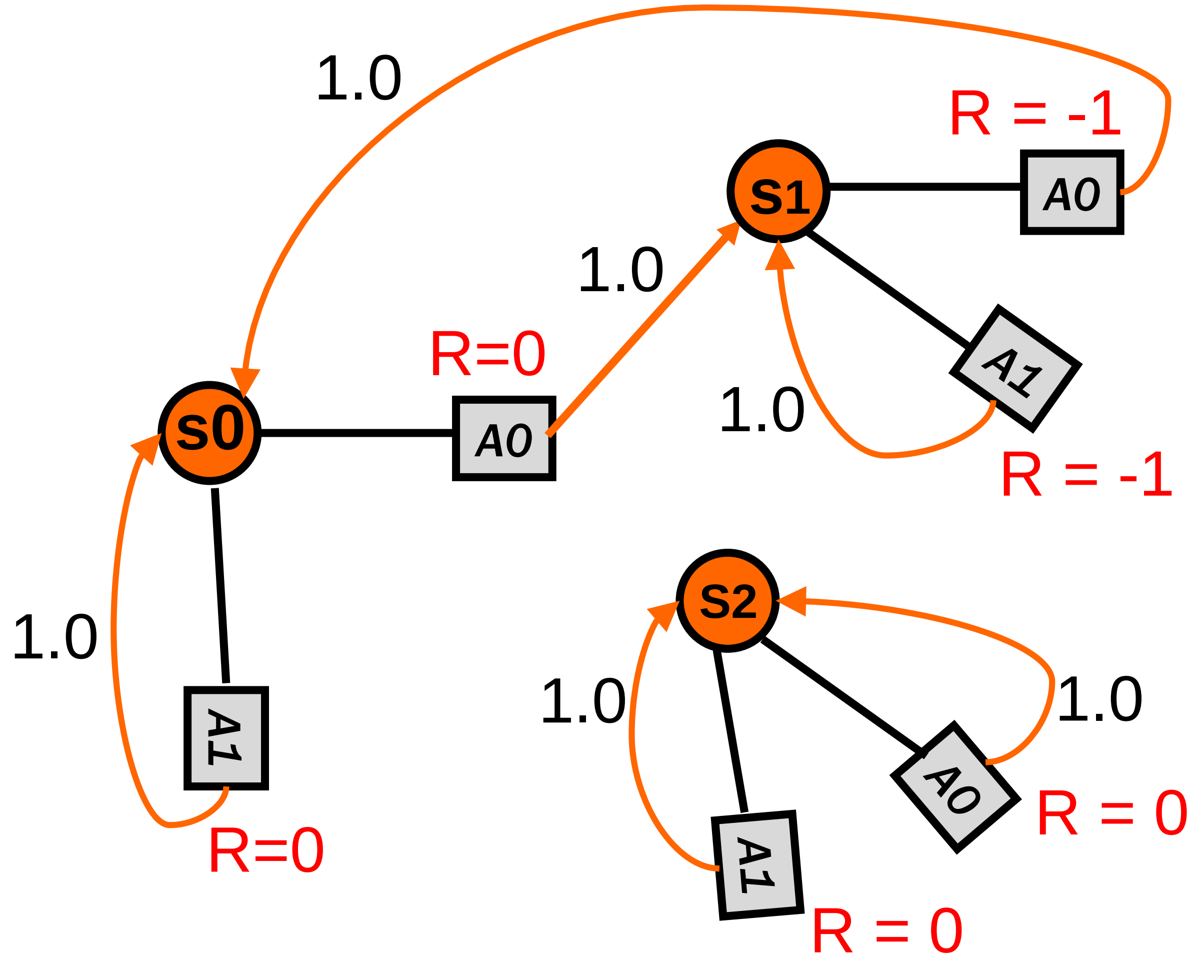
Initialization:

R=0

T(s,a,s)=1

Suppose our algorithm learns from s0,A1,s0,A0,s1,A0,s0

Revision of Naïve Approach



Learned MDP

Initialization:
R=0
T(s,a,s)=1

Optimal policy of learned MDP $\pi^*(s0) = A1$

Taking A1 in s0 provides no new info → policy will not improve

Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model
   (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

Will this work? No. Can get stuck in local minima.
(depends on initialization)

What can be done?

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

ADP-based (model-based) RL

1. Start with initial model
2. Solve for optimal policy given current model
(using value or policy iteration)
3. Take action according to an explore/exploit policy
(explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

Will this work? Depends on the explore/exploit policy.

Any ideas?

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

  - where V is current optimal value function estimate (based on current model), and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s') of the next state s'

- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence

Explore/Exploit Policies

- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but can be slow

- In practice it is common to simply set p(t) to a small constant ε (e.g. ε=0.1)
  - Called ε-greedy exploration
  - Just as we saw for bandits
  - ε usually set to small value (compared to 0.5) so the trajectories we learn from are mostly based on exploitation behavior

Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

The Impact of Temperature

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

- Suppose we have two actions and that Q(s,a1) = 1, Q(s,a2) = 2

- T=10 gives Pr(a1 | s) = 0.48, Pr(a2 | s) = 0.52
  - Almost equal probability, so will explore

- T= 1 gives Pr(a1 | s) = 0.27, Pr(a2 | s) = 0.73
  - Probabilities more skewed, so explore a1 less

- T = 0.25 gives Pr(a1 | s) = 0.02, Pr(a2 | s) = 0.98
  - Almost always exploit a2

Alternative Model-Based Approach: Optimistic Exploration

- There is a class of RL algorithms based on the idea of optimistic exploration.

- Basically, if the agent has not explored a state "enough", then it acts as if that state provides maximum reward
  - So actions will be selected to try and reach such states

- Many of the theoretical results are based on this idea
  - We'll only touch on the theory

Optimistic Exploration: Rmax Algorithm

1. Start with an optimistic model
   (assign largest possible reward to "unexplored states")
   (actions from "unexplored states" only self transition)

2. Solve for optimal policy in optimistic model (standard VI)

3. Take greedy action according to policy

4. Update optimistic estimated model
   (if a state becomes "known" then use its true statistics)

5. Goto 2

Agent always acts greedily according to a model that assumes all "unexplored" states are maximally rewarding

Rmax: Optimistic Model

- Let $N(s,a)$ be the number of times that action a has been tried in state s
  - A state-action pair (s,a) is "unexplored" if $N(s,a) < N_e$
  - A state s is unexplored if for some action a, (s,a) is unexplored

- **Optimistic Model Construction (only in the agents head):**
  - If $N(s,a) < N_e$ then $T(s,a,s) = 1$ and $R(s) = R_{\max}$
  - Unexplored states have max reward self loops

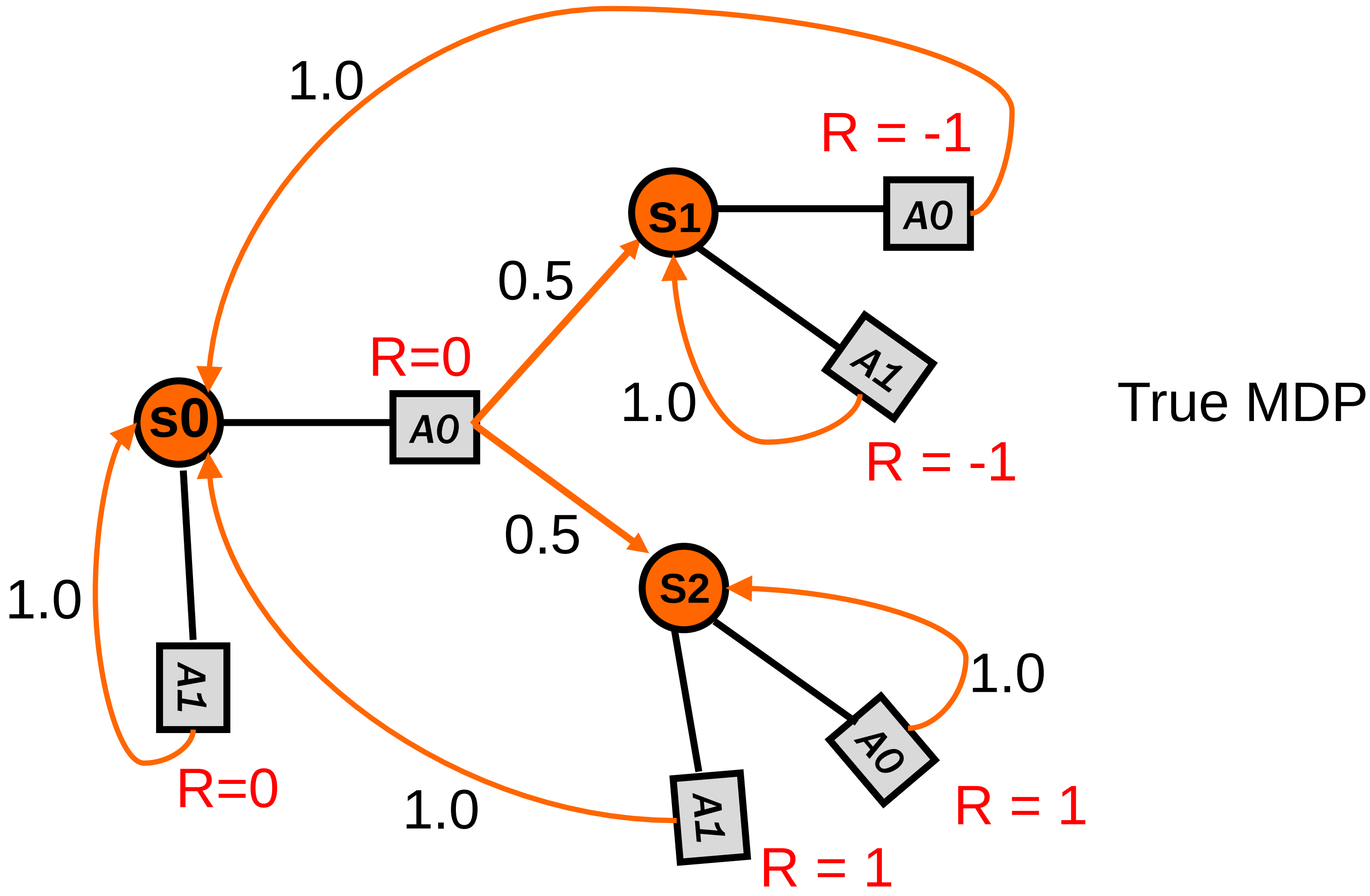
  - If $N(s,a) \geq N_e$ then $T(s,a,s')$ and $R(s)$ are estimated from the N_e experiences
  - Explored states are estimated from observed data

- For large enough N_e the explored states will have accurate models

Rmax: Optimistic Model

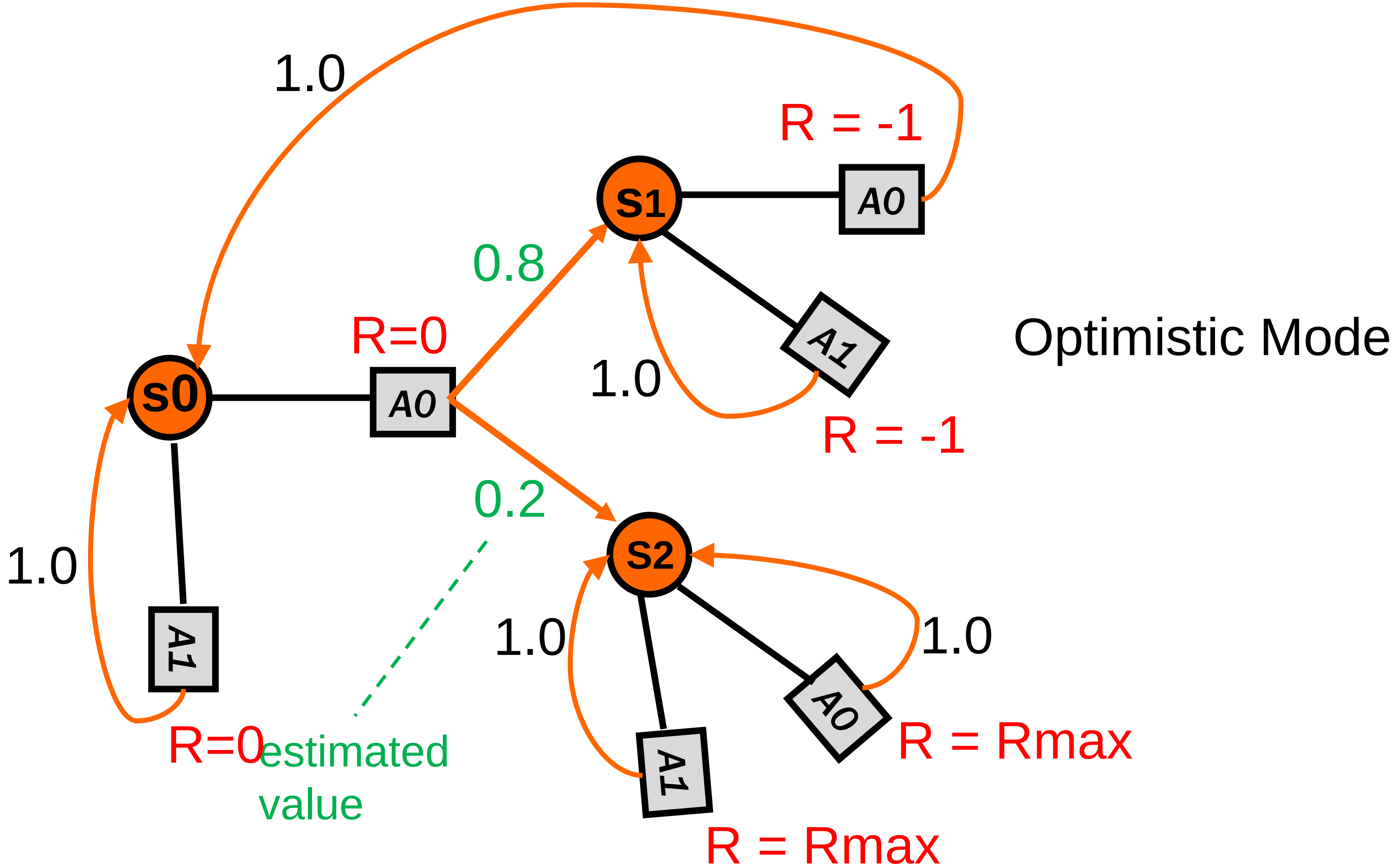
- **Optimistic Model Construction (only in the agents head):**
  - If $N(s,a) < N_e$ then $T(s,a,s) = 1$ and $R(s) = R_{\max}$
  - Unexplored states have max reward self loops
  - If $N(s,a) \geq N_e$ then $T(s,a,s')$ and $R(s)$ are estimated from the N_e experiences
  - Explored states are estimated from observed data
- An optimal policy for this optimistic model will try to reach unexplored states (those with unexplored actions) since it can stay at those states and accumulate maximum reward
- Never explicitly explores. Is always greedy, but with respect to an optimistic outlook.

Rmax Algorithm



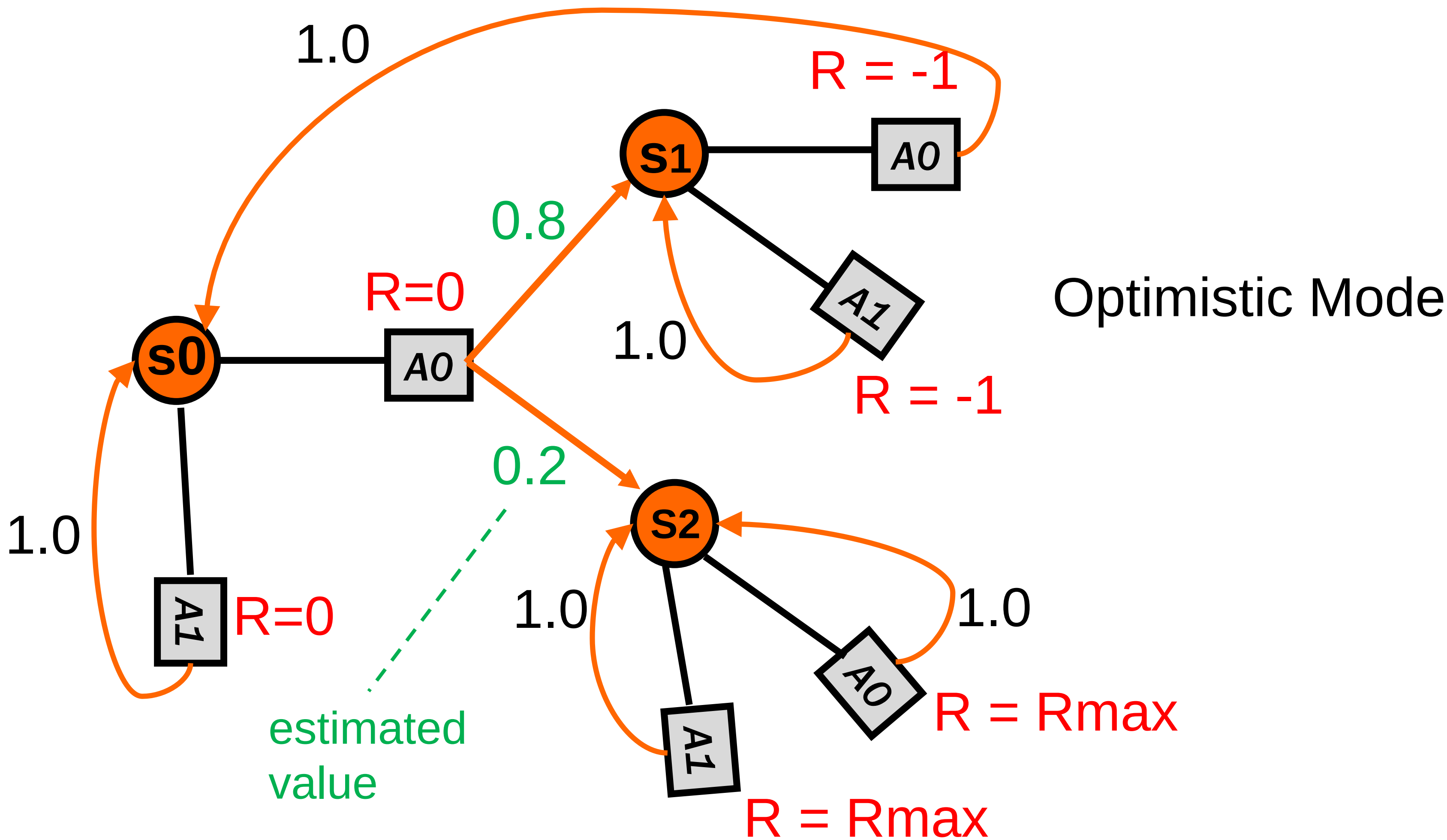
Suppose $N(s0, A0) = N(s0, A1) = N(s1, A0) = N(s1, A1) = N_e$

Revision of Naïve Approach



Suppose $N(s0, A0) = N(s0, A1) = N(s1, A0) = N(s1, A1) = N_e$

Revision of Naïve Approach



Optimistic policy will take A0 in s0 to try to reach s2

Will lead to eventually exploring s2 enough

Optimistic Exploration

- Is Rmax provably efficient?
  - If the model is ever completely learned (i.e. N(s,a) > N_e, for all (s,a), then the policy will be near optimal)
  - Recent results show that this will happen "quickly"

- **Theoretical Guarantee (Roughly speaking):** There is a value of N_e (depending on n,m, and Rmax), such that with high probability the Rmax algorithm will select at most a polynomial number of actions with value less than ε of optimal.

- RL can be solved in poly-time in n, m, and Rmax!

Why does the complexity depend on Rmax?

Good-bye Model Based

- So model-based methods have some strong theoretical guarantees
- But in practice they are difficult to use for large MDPs
  - Require storing and solving an estimated MDP model
- Some researchers are using model-based RL with planners for large MDPs (such as tree search)
  - A compact model is learned, e.g. a Dynamic Bayesian Network
- Most current practical applications of RL us model-free approaches
  - But, we might hypothesize that ultimately intelligent agents should maintain and use something that is “model like”