Symbolic Approach to MDP

- States represented as assignment to \( n \) (Boolean) variables
- Number of states is \( 2^n \)
- SPUDD: Main idea represent reward, transitions, and value function "symbolically"
- Implement dynamic programming algorithms "symbolically"

Decision Diagrams

- BDD representation of functions from \( \{0,1\}^n \to \{0,1\} \)
- ADD representation of functions from \( \{0,1\}^n \to \mathbb{R} \)
- BDD/ADD can be combined through binary operations (plus, mult, max, etc) using the Apply algorithm
- Calc: \( f_1 \cdot f_2, \ f_1^* f_2, \ max\{f_1, f_2\} \)

SPUDD

- Reward captured by ADD
- Transitions: DBN captures "2-time-slice" dependence of state variables before/after action.
- The conditional prob. \( p(x'_i = 1|x_1, \ldots, x_n, a) \) is captured by an ADD
- The conditional prob. \( p(x'_i | x_1, \ldots, x_n, a) \) is captured by the dual-action-diagram ADD

SPUDD

- The Bellman update
  \[
  V(s) = \max_a \left\{ r(s, a) + \gamma \sum_{s'} p(s'|s, a)V(s') \right\}
  \]
- Can be implemented symbolically
  \[
  V(x_1, x_2, \ldots, x_n) = \max_a \left\{ r(x_1, \ldots, x_n, a) + \gamma \sum_{x'_1, \ldots, x'_n} p(x'_1, \ldots, x'_n | x_1, \ldots, x_n, a)V(x'_1, \ldots, x'_n) \right\}
  \]
SPUDD

\[ V(x_1, x_2, \ldots, x_n) = \max_a \{ r(x_1, \ldots, x_n, a) + \gamma \sum_{x_1', x_2'} p(x_1', \ldots, x_n' | x_1, \ldots, x_n, a) V(x_1', \ldots, x_n') \} \]

And this expression can be evaluated more efficiently via

\[
\sum_{x_1} p(x_1' | x_1, \ldots, x_n, a) \sum_{x_2} p(x_2' | x_1, \ldots, x_n, a) \ldots \sum_{x_n} p(x_n' | x_1, \ldots, x_n, a) V(x_1', \ldots, x_n')
\]

Here assuming cond. Independence; can always be done
For some ordering of variables

Search in AI

• Search problem: state space and operators
• Un-informed search: BFS, DFS
• Iterative Deepening: combines benefits of both algorithms

• Informed search: admissible heuristic h
• A* algorithm: guided by h
• Extract h from problem relaxation

Search in AI

• And-Or search trees capture uncertainty about operator outcome (e.g. opponent move in games)
• Similar structure to MDP transitions where uncertainty is quantified probabilistically

LRTA* and RTDP

• LRTA*: one walk over the search space - improves/corrects h with every move
• Same structure as RTDP when RTDP is initialized with an upper bound on V (an “admissible” initialization) and when the policy is the greedy policy wrt V.

RTDP and LRTDP

• RTDP can get close to optimal V quickly but slow to fully converge
• Main issue is the greedy action choice
• Once the values of good actions converge we need not explore them further and can focus on possibly worse but non-converged portions

RTDP and LRTDP

• LRTDP labels states as solved if their Bellman residual (err in backup) is smaller than epsilon and if they can only reach solved states
• By avoiding solved states in exploration convergence can be faster
Bandits

- 3 types of bandit objectives
  - PAC and Simple Regret seek to identify a good/best arm after n steps
  - Cumulative Regret: seek to collect max reward within n steps

UCB algorithm: aims to optimize Cumulative Regret
- Automatic explore/exploit tradeoff by picking arm maximizing
  \[ Q(s, a) + \sqrt{\frac{2 \ln n}{n(s)}} \]
- Cumulative regret in n pulls is \( O(\log n) \)

Uniform bandits achieve PAC guarantee
- If using sufficiently large \( w \) then whp algorithm finds almost optimal arm
- Median elimination improves sampling cost

[theoretical bounds]
- UCB has "polynomially decaying" simple regret in terms of number of arm pulls
- Uniform bandits has "exponentially decaying" simple regret
- Epsilon-greedy bandits has (even more) "exponentially decaying" simple regret

Multi stage rollout
- Expensive: \( (kwh)^d \)
- Simulates \( d \) steps of policy improvement (cf. policy iteration)

Policy Rollouts
- On line planning:
  - Repeat:
    - optimize action just for the current state
    - Then take that action
  - Policy improvement for on line planning can be simulated via rollout
  - Use \( kwh \) simulation steps to make a decision
### Approximate Policy Iteration

- Simulate the process of policy iteration via rollouts and supervised learning
- Repeat:
  - Given current policy \( \pi \) generate trajectories of improve(\( \pi \)) by using the rollout procedure
  - Use supervised learning to learn a representation of the improved policy \( \pi' \)
  - \( \pi = \pi' \)

### Sparse Sampling

- Sparse sampling trees as an approximation of Expectimax tree
- Can be shown to have PAC guarantees (this time to get the optimal policy)
- But not reasonably efficient
- Can be seen as implementing a recursive set of uniform bandit algorithms
- Idea: same algorithm with non-uniform bandits

### Sparse Sampling

- SimQ*: a simulation of the depth h optimal Q value
- Can be implemented via recursive bandits
- The same as an "arm pull" at level h
  - Good algorithm
  - But bad anytime behavior
  - Not ideal for on line planning

### MCTS

- Grows a search tree incrementally
- At each stage
  - Pick actions within tree
  - Rollout baseline \( \pi \) (e.g. random) outside tree
- Each such trajectory provides an approximate Monte Carlo sample SimQ*(s,a,h) for each node in the tree
- This assumes that bandits at children provide optimal action and therefore the returned value sampled via SimQ*(s,a,h)

### MCTS

- Each such trajectory provides an approximate Monte Carlo sample SimQ*(s,a,h) for each node in the tree
- Therefore:
  - at node \( s \)
  - pick action \( a \), take transition,
  - [apply algorithm recursively] collect trajectory reward \( R \) and add current \( r \) to it
  - Update Q value as MC update with \( R \)
  - Return \( R \) to parent

### UCT

- An instance of MCTS where action selection within tree is done by UCB
- In the limit converges to optimal choice
- Practical successes
- Require good heuristic baseline or rollout policy
- Root vs internal nodes: different goals
- Can use different bandits