An Overview of MDP Planning Research at ICAPS

Andrey Kolobov and Mausam

Computer Science and Engineering
University of Washington, Seattle
What is ICAPS?

• International Conference on Automated Planning and Scheduling
  – 23rd conference: June 10-14, Rome, Italy.
  – 24th conference: June 21-26, Portsmouth, NH

• Traditional community: classical planning
• ~15 years: significant planning under uncertainty
  – Discrete MDPs
  – Discrete POMDPs
  – other kinds of uncertainty (e.g., disjunctive)
  – ~continuous state spaces
What is ICAPS?

• **International Conference on Automated Planning and Scheduling**
  – 23rd conference: June 10-14, Rome, Italy.
  – 24th conference: June 21-26, Portsmouth, NH

• **Traditional community: classical planning**

• **~15 years: significant planning under uncertainty**
  – Discrete MDPs
  – Discrete POMDPs
  – other kinds of uncertainty (e.g., disjunctive)
  – ~continuous state spaces
International Probabilistic Planning Competition


• Salience
  – shapes the research in the community

• Slightly controversial history
  – ~deterministic planners beat MDP planners (‘04, ‘06)
  – how to represent a problem: PPDDL vs. RDDL (‘11)
Strength/Bias of AI Planning

- Causal planning
- Temporal planning
- Resource planning
- Spatial planning
- ...
Does Planning Want to be Slow?

- **In theory**
  - yes (classical planning in STRIPS is PSPACE)
  - YES (MDP planning in factored problems in EXPTIME)

- **In practice**
  - NO

- It is what you make it out to be
  - optimal
  - quality bound
  - time bound
  - anytime
  - interleaved planning and execution
  - ...

3 Key Messages

• **M#0: No need for exploration-exploitation tradeoff**
  – planning is purely a computational problem (V.I. vs. Q)

• **M#1: Search in planning**
  – states can be ignored or reordered for efficient computation

• **M#2: Representation in planning**
  – develop interesting representations for Factored MDPs
    → Exploit structure to design domain-independent algorithms

• **M#3: Goal-directed MDPs**
  – design algorithms that use explicit knowledge of goals
Outline

• Introduction

• MDP Model

• Uninformed Algorithms

• Heuristic Search Algorithms

• Approximation Algorithms

• Extension of MDPs
Shameless Plug
Outline

• Introduction
• MDP Model
• Uninformed Algorithms
• Heuristic Search Algorithms
• Approximation Algorithms
• Extension of MDPs
Stochastic Shortest-Path MDPs

[ Bertsekas 1995 ]

SSP MDP is a tuple $< S, A, T, C, G >$, where:

- $S$ is a finite state space
- $A$ is a finite action set
- $T: S \times A \times S \rightarrow [0, 1]$ is a stationary transition function
- $C: S \times A \times S \rightarrow \mathbb{R}$ is a stationary cost function ($= -R: S \times A \times S \rightarrow \mathbb{R}$)
- $G$ is a set of absorbing cost-free goal states

Under two conditions:

- There is a proper policy (reaches a goal with $P=1$ from all states)
- Every improper policy incurs a cost of $\infty$ from every state from which it does not reach the goal with $P=1$
SSP MDP Objective Function

• Find policy $\pi : S \rightarrow A$
• Minimize expected (undiscounted) cost to reach a goal
  – agent acts for an indefinite horizon

• A cost-minimizing policy is always proper!
Bellman Equations

\[
V^*(s) = \begin{cases} 
0 & \text{if } s \in \mathcal{G} \\
\min_{a \in A} \sum_{s' \in S} T(s, a, s') \left[ C(s, a, s') + V^*(s') \right] & \text{otherwise}
\end{cases}
\]

\[Q^*(s, a)\]

\[V^*(s) = \min_a Q^*(s, a)\]
Not an SSP MDP Example

No cost-free "loops" allowed!

C(s₁, a₂, s₁) = 7.2
C(s₁, a₁, s₂) = 1
C(s₂, a₁, s₁) = -1
C(s₂, a₂, s₂) = -3
T(s₂, a₂, s₃) = 0.3
C(s₂, a₂, s₃) = 1
C(s₃, a₁, s₃) = 2.4
C(s₃, a₂, s₃) = 0.8
C(s₃, a₀, s₃) = 0
C(s₃, a₁, s₃) = 0
C(s₃, a₃, s₃) = 0
C(s₃, a₄, s₃) = 0

No dead ends allowed!
SSP and Other MDP Classes

- Infinite Horizon Discounted Reward MDPs
- Finite Horizon MDPs

• Will concentrate on SSP in the rest of the tutorial
Discounted Reward MDP $\rightarrow$ SSP

[Bertsekas&Tsitsiklis 95]
Outline

• Introduction
• MDP Model
  • Uninformed Algorithms
  • Heuristic Search Algorithms
  • Approximation Algorithms
• Extension of MDPs
Algorithms

- Value Iteration
- Policy Iteration
- Asynchronous Value Iteration
  - Prioritized Sweeping
    - Many priority functions (some better for goal-directed MDPs)
  - Backward Value Iteration: priorities w/o priority queue
  - Partitioned Value Iteration
    - Topological Value Iteration
  - ...
- Linear Programming
**Backward VI** [Dai&Hansen 07]

- **Prioritized Sweeping has priority**
  - Prioritized VI without priority queue

- **Backup states in reverse order starting from goal**
  - don’t repeat a state in an iteration
  - other optimizations
  - (backup only states in current greedy subgraph)

- **Characteristics**
  - no overhead of priority queue
  - good information flow
(General) Partitioned VI

1. Initialize $V$ arbitrarily
2. Construct a partitioning of states $\mathcal{P} = \{p_i\}$
3. (Optional) Initialize priorities for each $p_i$
4. Repeat
   5. Select a partition $p'$
   6. Perform (potentially several) backups for all states in $p'$
   7. (Optional) Update priorities for all predecessor partitions of $p'$
5. Until termination;
6. Return greedy policy $\pi^V$

How to construct a partition?
How many backups to perform per partition?
How to construct priorities?
Topological VI [Dai & Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to $\epsilon$-consistency

M#1: states can be reordered for efficient computation
Other Benefits of Partitioning

• **External-memory algorithms**
  – PEMVI [Dai et al 08, 09]
    • partitions live on disk
    • get each partition from the disk and backup all states

• **Cache-efficient algorithms**
  – P-EVA algorithm [Wingate&Seppi 04a]

• **Parallelized algorithms**
  – P3VI (Partitioned, Prioritized, Parallel VI) [Wingate&Seppi 04b]
Outline of the Tutorial

• Introduction
• MDP Model
• Uninformed Algorithms
• Heuristic Search Algorithms
• Approximation Algorithms
• Extension of MDPs
Heuristic Search

• **Insight 1**
  – knowledge of a start state to save on computation
    ~ (all sources shortest path $\Rightarrow$ single source shortest path)

• **Insight 2**
  – additional knowledge in the form of heuristic function
    ~ (dfs/bfs $\Rightarrow$ A*)
Model

• SSP (as before) with an additional start state $s_0$
  – denoted by $\text{SSP}_{s_0}$

• Solution to an SSP$_{s_0}$: *partial policy closed w.r.t. $s_0$*
  – defined for all states $s'$ reachable by $\pi_{s_0}$ starting from $s_0$
  – many states are irrelevant (e.g., not reachable from $s_0$)
Heuristic Function

• $h(s): S \rightarrow \mathbb{R}$
  – estimates $V^*(s)$
  – gives an indication about “goodness” of a state
  – usually used in initialization $V_0(s) = h(s)$
  – helps us avoid seemingly bad states

• Define *admissible* heuristic
  – optimistic
  – $h(s) \leq V^*(s)$
A General Scheme for Heuristic Search in MDPs

• **Two (over)simplified intuitions**
  – Focus on states in greedy policy wrt $V$ rooted at $s_0$
  – Focus on states with residual $> \epsilon$

• **Find & Revise**: [Bonet&Geffner 03]
  – repeat
    • find a state that satisfies the two properties above
    • perform a Bellman backup
  – until no such state remains

• **Convergence to $V^*$ is guaranteed**
  – if heuristic function is admissible
  – no state in greedy policy gets starved in $\infty$ FIND steps
A* $\rightarrow$ LAO*

regular graph

soln: (shortest) path

A*

acyclic AND/OR graph

soln: (expected shortest) acyclic graph

AO* [Nilsson’71]

cyclic AND/OR graph

soln: (expected shortest) cyclic graph

LAO* [Hansen&Zil.’98]

All algorithms able to make effective use of reachability information!
LAO*

\[ V(s_0) = h(s_0) \]

Add \( s_0 \) to the fringe and in greedy graph.
LAO*

FIND: expand best state s on the fringe (in greedy graph)

\[ V(s_0) = h(s_0) \]
FIND: expand best state $s$ on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach $s$
perform VI on this subset
FIND: expand best state \(s\) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \(s\)
perform VI on this subset
recompute the greedy graph
**LAO**

FIND: expand best state $s$ on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach $s$
perform VI on this subset
recompute the greedy graph
FIND: expand best state \( s \) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \( s \)
perform VI on this subset
recompute the greedy graph
FIND: expand best state $s$ on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach $s$
perform VI on this subset
recompute the greedy graph
FIND: expand best state \( s \) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \( s \)
perform VI on this subset
recompute the greedy graph
FIND: expand best state $s$ on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach $s$
perform VI on this subset
recompute the greedy graph
FIND: expand best state \( s \) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \( s \)
perform VI on this subset
recompute the greedy graph
FIND: expand best state \( s \) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \( s \)
perform VI on this subset
recompute the greedy graph
FIND: expand best state $s$ on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach $s$
perform VI on this subset
recompute the greedy graph
FIND: expand best state \( s \) on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach \( s \)
perform VI on this subset
recompute the greedy graph
**LAO***

**FIND:** expand best state s on the fringe (in greedy graph)
initialize all new states by their heuristic value
subset = all states in expanded graph that can reach s
perform VI on this subset
recompute the greedy graph
output the greedy graph as the final policy
output the greedy graph as the final policy
M#1: some states can be ignored for efficient computation

$s_4$ was never expanded
$s_8$ was never touched
Real Time Dynamic Programming
[Barto et al 95]

• **Original Motivation**
  – agent acting in the real world

• **Trial**
  – simulate greedy policy starting from start state;
  – perform Bellman backup on visited states
  – stop when you hit the goal

• **RTDP: repeat trials forever**
  – Converges in the limit #trials → ∞
Trial
Trial

start at start state
repeat
perform a Bellman backup
simulate greedy action
Trial

start at start state
repeat
  perform a Bellman backup
  simulate greedy action
Trial

\[
\text{start at start state} \quad \text{repeat} \\
\quad \text{perform a Bellman backup} \\
\quad \text{simulate greedy action}
\]
Trial

start at start state
repeat

perform a Bellman backup
simulate greedy action
Trial

start at start state
repeat

perform a Bellman backup
simulate greedy action
Trial

start at start state
repeat
perform a Bellman backup
simulate greedy action
until hit the goal
RTDP
repeat forever

start at start state
repeat
perform a Bellman backup
simulate greedy action
until hit the goal
Real Time Dynamic Programming

[Barto et al 95]

• **Original Motivation**
  – agent acting in the real world

• **Trial**
  – simulate greedy policy starting from start state;
  – perform Bellman backup on visited states
  – stop when you hit the goal

• **RTDP: repeat trials forever**
  – Converges in the limit #trials → ∞
Termination Test Take 1: Labeling

[Bonet&Geffner 03b]

- Admissible heuristic & monotonicity
  \[ V(s) \leq V^*(s) \]
  \[ Q(s,a) \leq Q^*(s,a) \]

- Label a state \( s \) as solved
  - if \( V(s) \) has converged

\[ \text{Res}^V(s) < \epsilon \]
\[ \Rightarrow V(s) \text{ won't change!} \]
\[ \Rightarrow \text{label } s \text{ as solved} \]
$\text{Labeling (contd)}$

$\text{Res}^V(s) < \epsilon$  
$s'$ already solved  
$\Rightarrow V(s)$ won’t change!

label $s$ as solved
Labeling (contd)

M#3: some algorithms use explicit knowledge of goals

\[ V(s') \geq V(s) \quad \Rightarrow \quad V(s) \text{ won’t change!} \]

label s as solved

M#1: some states can be ignored for efficient computation

\[ V(s') \geq V(s) \quad \Rightarrow \quad V(s) \text{ won’t change!} \]

label s, s’ as solved

\[ \text{Res}^V(s) < \epsilon \quad \Rightarrow \quad s' \text{ already solved} \]
Labeled RTDP [Bonet&Geffner 03b]

repeat
    \( s \leftarrow s_0 \)
    label all goal states as solved
repeat //trials
    REVISE \( s \); identify \( a_{\text{greedy}} \)
    FIND: sample \( s' \) from \( T(s, a_{\text{greedy}}, s') \)
    \( s \leftarrow s' \)
until \( s \) is solved

for all states \( s \) in the trial
    try to label \( s \) as solved
until \( s_0 \) is solved

M#3: some algorithms use explicit knowledge of goals

M#1: some states can be ignored for efficient computation
LRTDP Extensions

• Different ways to pick next state
• Different termination conditions

• Bounded RTDP [McMahan et al 05]
• Focused RTDP [Smith&Simmons 06]
• Value of Perfect Information RTDP [Sanner et al 09]
Action Elimination

If $Q_l(s,a_1) > V_u(s)$ then $a_1$ cannot be optimal for $s$. 
Topological VI [Dai&Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to $\epsilon$-consistency: reverse top. order
Topological VI [Dai&Goldsmith 07]

- Identify strongly-connected components
- Perform topological sort of partitions
- Backup partitions to $\epsilon$-consistency: reverse top. order
Focused Topological VI \[\text{[Dai et al 09]}\]

- **Topological VI**
  - hopes there are many small connected components
  - can’t handle reversible domains...

- **FTVI**
  - initializes $V_l$ and $V_u$
  - LAO*-style iterations to update $V_l$ and $V_u$
  - eliminates actions using action-elimination
  - Runs TVI on the resulting graph
Where do Heuristics come from?

• Domain-dependent heuristics

• Domain-independent heuristics
  – dependent on specific domain representation

M#2: factored representations expose useful problem structure
Outline of the Tutorial

- Introduction
- MDP Model
- Uninformed Algorithms
- Heuristic Search Algorithms
- Approximation Algorithms
- Extension of MDPs
Symbolic Representations

• **Use of Algebraic Decision Diagrams for MDPs**
  – SPUDD (Symbolic VI) [Hoey et al 99]
  – Symbolic LAO* [Feng & Hansen 02]
  – Symbolic RTDP [Feng & Hansen 03]

• XADDs [Sanner et al 11]

• FODDs [Joshi et al 10]
Outline of the Tutorial

• Introduction
• MDP Model
• Uninformed Algorithms
• Heuristic Search Algorithms
• Approximation Algorithms
• Extension of MDPs
Motivation

• Even $\pi^*$ closed wr.t. $s_0$ is often too large to fit in memory...

• ... and/or too slow to compute ...

• ... for MDPs with complicated characteristics
  – Large branching factors/high-entropy transition function
  – Large distance to goal
  – Etc.

• Must sacrifice optimality to get a “good enough” solution
Overview

Online
- Determinization-based techniques
- Monte-Carlo planning

Offline
- Heuristic search with inadmissible heuristics
- Dimensionality reduction
- Hierarchical planning
- Hybridized planning
Overview

• Not a “golden standard” classification
  – In some aspects, arguable
  – Others possible, e.g., optimal vs. suboptimal in the limit

• Approaches differ in the quality aspect they sacrifice
  – Probability of reaching the goal
  – Expected cost of reaching the goal
  – Both
Approximation Algorithms

✓ Overview

• Determinization-based Algorithms
• Heuristic Search with Inadmissible Heuristics
• Dimensionality Reduction
• Monte-Carlo Planning
• Hierarchical Planning
• Hybridized Planning
Determinization-based Techniques

• High-level idea:
  1. Compile MDP into its *determinization*
  2. Generate plans in the determinization
  3. Use the plans to choose an action in the curr. state
  4. Execute, repeat

• Key insight:
  – Classical planners are *very fast*

• Key assumption:
  – The MDP is goal-oriented in *factored representation*
Detour: Factored MDPs

• How to describe an MDP instance?
  – $S = \{s_1, \ldots, s_n\}$ — flat representation
  – $T(s_i, a_j, s_k) = p_{i,j,k}$ for every state, action, state triplet
  – ...

• Flat representation too cumbersome!
  – Real MDPs have billions of billions of states
  – Can’t enumerate transition function explicitly

• Flat representation too uninformative!
  – State space has no meaningful distance measure
  – Tabulated transition/reward function doesn’t expose structure
Detour: Example Factored MDP

- $S = \{(x_1, \ldots, x_n) \mid x_1 \text{ in } X_1, \ldots, x_n \text{ in } X_n\}$

- $A = \ldots$

- $T = 0.3 \quad 0.7$

- $C = \ldots$

- Effects

- Precondition
Back to Determinization-based Planning: All-Outcome Determinization

Each outcome of each probabilistic action $\rightarrow$ separate action
Most-Likely-Outcome Determinization
FF-Replan: Overview & Example

1) Find a goal plan in a determinization
2) Try executing it in the original MDP
3) Replan & repeat if unexpected outcome

[Yoon, Fern, Givan, 2007]
FF-Replan: Theoretical Properties

- Optimizes the *MAXPROB* criterion — $P_G$ of reaching the goal
  - In SSPs, this is always 1.0 — FF-Replan always tries to avoid cycles!
  - Super-efficient on SSPs w/o dead ends
  - Won two IPPCs (-2004 and -2006) against much more sophisticated techniques

- Ignores probability of deviation from the found plan
  - Results in long-winded paths to the goal
  - Troubled by *probabilistically interesting MDPs* [Little, Thiebaux, 2007]
    - There, an unexpected outcome may lead to catastrophic consequences

- In particular, breaks down in the presence of dead ends
  - Originally designed for MDPs without them
FF-Hindsight: Putting “Probabilistic” Back Into Planning

• FF-Replan is oblivious to probabilities
  – Its main undoing
  – How do we take them into account?

• **Suppose you knew the future**
  – Knew would happen if you executed action $a$ $t$ steps from now
  – The problem would be deterministic, could easily solve it

• **Don’t actually know the future...**
  – ...but can sample several futures probabilistically
  – Solutions tell which actions are likely to start successful policies

• Basic idea behind **FF-Hindsight**
Objective: Optimize MAXPROB criterion

Time 1

Action
Probabilistic Outcome

Left Outcomes are more likely

Time 2

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
FF-Hindsight: Sampling a Future-1

Maximize Goal Achievement

Action

Probabilistic Outcome

Time 1

Time 2

A1: 1
A2: 0

Dead End
Goal State

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
FF-Hindsight: Sampling a Future-2

Maximize Goal Achievement

Time 1

A1
A2

Time 2

A1
A2

Action
Probabilistic Outcome

A1: 2
A2: 1

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
Providing Solution Guarantees

• FF-Replan provides no solution guarantees
  – May have $P_G = 0$ on SSPs with dead ends, even if $P^*_G > 0$

• FF-Hindsight provides only theoretical guarantees
  – Practical implementations too distinct from theory

• RFF (Robust FF) provides quality guarantees in practice
  – Constructs a policy tree out of deterministic plans
1. Generate either the AO or MLO determinization. Start with the policy graph consisting of the initial state $s_0$ and all goal states $G$. 

$S_0$ $G$
2. Run FF on the chosen determinization and add all the states along the found plan to the policy graph.
RFF: Adding Alternative Outcomes

3. Augment the graph with states to which other outcomes of the actions in the found plan could lead and that are not in the graph already. They are the policy graph’s *fringe states*. 
4. Run VI to propagate heuristic values of the newly added states. This possibly changes the graph’s fringe and helps avoid dead ends!
5. Estimate the probability $P(\text{failure})$ of reaching the fringe states (e.g., using Monte-Carlo sampling) from $s_0$. This is the current partial policy’s failure probability w.r.t. $s_0$. If $P(\text{failure}) > \epsilon$ Else, done!
6. From each of the fringe states, run FF to find a plan to reach the goal or one of the states already in the policy graph.

Go back to step 3: Adding Alternative Outcomes
Summary of Determinization Approaches

• Revolutionized SSP MDPs approximation techniques
  – Harnessed the speed of classical planners
  – Eventually, started to take into account probabilities
  – Help optimize for a “proxy” criterion, MAXPROB

• Classical planners help by quickly finding paths to a goal
  – Takes “probabilistic” MDP solvers a while to find them on their own

• However...
  – Still almost completely disregard expect cost of a solution
  – Often assume uniform action costs (since many classical planners do)
  – So far, not useful on FH and IHDR MDPs turned into SSPs
    • Reaching a goal in them is trivial, need to approximate reward more directly
  – Impractical on problems with large numbers of outcomes
Approximation Algorithms

✓ Overview
✓ Determinization-based Algorithms
  • Heuristic Search with Inadmissible Heuristics
  • Dimensionality Reduction
  • Monte-Carlo Planning
  • Hierarchical Planning
  • Hybridized Planning
The FF Heuristic

- Taken directly from deterministic planning
- Uses the all-outcome determinization of an MDP
  - But ignores the *delete effects* (negative literals in action outcomes)

\[ h_{FF}(s) = \text{approximately optimal cost of a plan from } s \text{ to a goal in the delete relaxation} \]

- Very fast due to using the delete relaxation
- Very informative

[Hoffmann and Nebel, 2001]
The GOTH Heuristic

[Kolobov, Mausam, Weld, 2010a]

• Designed for MDPs at the start (not adapted classical)

• Motivation: would be good to estimate $h(s)$ as cost of a non-relaxed deterministic goal plan from $s$
  – But too expensive to call a classical planner from every $s$
  – Instead, call from only a few $s$ and generalize estimates to others

• More details in Mausam’s talk...
Approximation Algorithms

✓ Overview
✓ Determinization-based Algorithms
✓ Heuristic Search with Inadmissible Heuristics

• Dimensionality Reduction
• Monte-Carlo Planning
• Hierarchical Planning
• Hybridized Planning
Dimensionality Reduction: Motivation

• No approximate methods so far explicitly try to save space

• Dimensionality reduction attempts to do exactly that
  – Insight: $V^*$ and $\pi^*$ are functions of $\sim |S|$ parameters (states)
  – Replace it with an approximation with $r \ll |S|$ params ...
  – ... in order to save space

• How to do it?
  – Factored representations are crucial for this
  – View $V$ & $\pi$ as functions of state variables, not states themselves
  – Obtain $r$ basis functions $b_i$, let $V^*(s) \approx \sum_i w_i b_i(s)$, find $w_i$ s
ReTrASE

• Derives *basis functions* from the all-outcomes determinization

• Sets $V(s) = \min_i w_i B_i(s)$

• Learns weights $w_i$ by running an RTDP-like procedure

• More details in Mausam’s talk...
Approximate LP

Let $V^*(s) \approx \sum_i w_i b_i(s)$, where $b_i$'s are given by the user

Insight: assume each $b$ depends on at most $z \ll |X|$ vars

Then, can formulate LP with only $O(2^z)$ constraints to find $w_i$'s
- Much smaller than $|S| = O(2^{|X|})$
FPG

- Directly learns a policy, not a value function
- For each action, defines a *desirability function*

$\theta_{a,1}, \ldots, \theta_{a,m}$ for each $a$

- Mapping from state variable values to action “quality”
  - Represented as a neural network
  - Parameters to learn are network weights $\theta_{a,1}, \ldots, \theta_{a,m}$ for each $a$
FPG

• Policy (distribution over actions) is given by a softmax

\[
\pi_{\tilde{\theta}}(s, a) = \frac{e^{f_{\alpha|\tilde{\theta}_\alpha}(s)}}{\sum_{a' \in A} e^{f_{\alpha'|\tilde{\theta}_{\alpha'}}(s)}},
\]

• To learn the parameters:
  – Run trials (similar to RTDP)
  – After taking each action, compute the gradient w.r.t. weights
  – Adjust weights in the direction of the gradient
  – Makes actions causing expensive trajectories to be less desirable
Approximation Algorithms

✓ Overview
✓ Determinization-based Algorithms
✓ Heuristic Search with Inadmissible Heuristics
✓ Dimensionality Reduction
  • Monte-Carlo Planning
  • Hierarchical Planning
  • Hybridized Planning
What Planning Approaches Couldn’t Do (Until Recently)

• The *Sysadmin* problem:

\[
A: \quad R: \sum_i [\text{Ser}_i = \uparrow] \quad T:
\]

\[
\begin{align*}
\text{Time } t-1 & \quad \text{Time } t \\
\text{Restart(Ser}_1) \quad & \quad \text{P(Ser}_1^t \mid \text{Restart}^{t-1}(\text{Ser}_1), \text{ Ser}_1^{t-1}, \text{ Ser}_2^{t-1}) \\
\text{Restart(Ser}_2) \quad & \quad \text{P(Ser}_2^t \mid \text{Restart}^{t-1}(\text{Ser}_2), \text{ Ser}_1^{t-1}, \text{ Ser}_2^{t-1}, \text{ Ser}_3^{t-1}) \\
\text{Restart(Ser}_3) \quad & \quad \text{P(Ser}_3^t \mid \text{Restart}^{t-1}(\text{Ser}_3), \text{ Ser}_2^{t-1}, \text{ Ser}_3^{t-1})
\end{align*}
\]
Monte-Carlo Planning: Motivation

• Characteristics of Sysadmin:
  – FH MDP turned $SSP_{s_0}$ MDP
    • Reaching the goal is trivial, determinization approaches not really helpful
  – Enormous reachable state space
  – High-entropy $T$ ($2^{|X|}$ outcomes per action, many likely ones)
    • Building determinizations can be super-expensive
    • Doing Bellman backups can be super-expensive

• Solution: *Monte-Carlo Tree Search*
  – Does not manipulate $T$ or $C/R$ explicitly – no Bellman backups
  – Relies on a *world simulator* – indep. of MDP description size
UCT: A Monte-Carlo Planning Algorithm

• **UCT** [Kocsis & Szepesvari, 2006] computes a solution by simulating the current best policy and improving it
  – Similar principle as RTDP
  – But action selection, value updates, and guarantees are different

• **Success stories:**
  – Go (thought impossible in ‘05, human grandmaster level at 9x9 in ‘08)
  – Klondike Solitaire (wins 40% of games)
  – General Game Playing Competition
  – Real-Time Strategy Games
  – Probabilistic Planning Competition (-2011)
  – The list is growing...
When all node actions tried once, select action according to tree policy

$\pi_{UCT}(s) = \arg \max_a Q(s, a) + c \sqrt{\frac{\ln n(s)}{n(s, a)}}$
UCT Details

• Rollout policy:
  – Basic UCT uses random

• Tree policy:
  – $Q'(s,a)$: average reward received in current trajectories after taking action $a$ in state $s$
  – $n(s,a)$: number of times action $a$ taken in $s$
  – $n(s)$: number of times state $s$ encountered

$$\pi_{UCT}(s) = \arg\max_a Q'(s, a) + c \sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Exploration term
UCT Caveat

• Optimizes *cumulative regret*
  – Total amount of regret during state space exploration
  – Appropriate in RL

• In planning, *simple regret* is more appropriate
  – Regret during state space exploration is fictitious
  – Only regret of the final policy counts!
BRUE: MCTS for Planning
[Feldman and Domshlak, 2012]

• Modifies UCT to minimize simple regret directly

• In UCT, simple regret diminishes at a polynomial rate in # samples

• In BRUE, simple regret diminishes at an exponential rate
Summary

• Surveyed 4 different approximation families
  – Dimensionality reduction
  – Inadmissible heuristic search
  – Dimensionality reduction
  – MCTS
    – Hierarchical planning
    – Hybridized planning

• Sacrifice different solution quality aspects

• Lots of work to be done in each of these areas
Beyond Stochastic Shortest-Path MDPs

[110]

-Kolobov, Mausam, Weld, 2012

-But real problems do have dead ends!
-What does the “best” policy mean in the presence of dead ends? How do we find it?
- More details in the talk on pitfalls of goal-oriented MDPs

No dead ends allowed in SSP!
Conclusion: 3 Key Messages

• **M#0: No need for exploration-exploitation tradeoff**
  – Planning is purely a computational problem
  – VI, PI

• **M#1: Search in planning**
  – States can be ignored or reordered for efficient computation
  – TVI

• **M#2: Representation in planning**
  – Efficient planning thanks to informative representations
  – Det.-based planning, dimensionality reduction

• **M#3: Goal-directed MDPs**
  – Design algorithms that use explicit knowledge of goals
  – Det.-based planning, heuristic search