Model-based Probabilistic Planning

Mausam

Computer Science and Engineering
Indian Institute of Technology (IIT) Delhi
Goal

a brief introduction to various goal-oriented MDPs
an extensive discussion of heuristic search
a discussion on connections with classical planning
Plan

• **Lecture 10**
  – Definition of Stochastic Shortest Path
  – Various Heuristic Search Algorithms for SSPs

• **Lecture 11**
  – Extensions of SSPs for MDPs with Dead Ends
  – Determinization-based Approximation Algorithms
Our Setting

- vs. RL (Zico): model of the world is known
- vs. flat: model of the world in a declarative representation
  - symbolic
  - large problems
- vs. reward (Scott): goal directed
  - PPDDL vs RDDL
- vs. finite-horizon MDPs (Thomas): indefinite horizon
- vs. classical planning (Malte/Gabi): probabilities
- vs. complete state space: knowledge of the start state
- domain independent: no additional human input
Heuristic Search for SSPs

Mausam

Computer Science and Engineering
Indian Institute of Technology (IIT) Delhi
Infinite Horizon Discounted Reward MDP

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,s')**: transition model
- **R(s,a,s')**: reward
- **γ**: discount factor
Where Does $\gamma$ Come From?

- $\gamma$ can affect optimal policy significantly
  - $\gamma = 0 + \varepsilon$: yields myopic policies for “impatient” agents
  - $\gamma = 1 - \varepsilon$: yields far-sighted policies, inefficient to compute

- How to set it?
  - Sometimes suggested by data
    - (e.g., inflation or interest rate)
  - Often set to whatever gives a reasonable policy
Infinite Horizon Discounted Reward MDP

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,s')**: transition model
- **R(s,a,s')**: reward
- **γ**: discount factor
Stochastic Shortest Path MDP

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,s’)**: Transition model
- **R(s,a,s’)**: Reward
- **γ**: Discount factor
Stochastic Shortest Path MDP

- **S**: A set of states
- **A**: A set of actions
- **T\((s,a,s')\)**: transition model
- **C\((s,a,s')\)**: cost
- **\(\gamma\)**: discount factor
Stochastic Shortest Path MDP

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,s')**: transition model
- **C(s,a,s')**: cost
Stochastic Shortest Path MDP

- **$S$:** A set of states
- **$A$:** A set of actions
- **$T(s,a,s')$:** transition model
- **$C(s,a,s')$:** cost
- **$G$:** set of goals

Minimize
- expected cost to reach a goal
- under full observability
- indefinite horizon
Bellman Equations for SSP

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

add base case; no discount factor
SSP vs. IHDR?

Discounted-reward MDPs

SSP

Finite-horizon MDPs
Discounted Reward MDP $\rightarrow$ SSP

[Bertsekas&Tsitsiklis 95]
When is SSP well formed/defined

[ Bertsekas, 1995 ]

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,sʼ)**: transition model
- **C(s,a,sʼ)**: cost
- **G**: set of goals

**Under two conditions:**

- There is a *proper policy* (reaches a goal with P=1 from all states)
- Every *improper policy* incurs a cost of ∞ from every state from which it does not reach the goal with P=1
Not an SSP MDP Example

No dead ends allowed!
Not an SSP MDP Example

No cost-free “loops” allowed!

- $C(s_1, a_2, s_1) = 7.2$
- $C(s_1, a_1, s_2) = 1$
- $C(s_2, a_1, s_1) = -1$

No dead ends allowed!

- $C(s_2, a_2, s_2) = -3$
- $T(s_2, a_2, s_G) = 0.7$
- $C(s_G, a_2, s_G) = 0$
- $C(s_G, a_1, s_G) = 0$
Value Iteration [Bellman 57]

No restriction on initial value function

1. Initialize $V_0$ arbitrarily for each state
2. $n \leftarrow 0$
3. Repeat
   4. $n \leftarrow n + 1$
   5. For each $s \in S$ do
      6. Compute $V_n(s)$ using Bellman backup at $s$
      7. Compute residual $r_n(s) = |V_n(s) - V_{n-1}(s)|$
   8. End
5. Until $\max_{s \in S} r_n(s) < \epsilon$
6. Return greedy policy: $\pi^V_n(s) = \arg\min_{a \in A} \sum_{s' \in S} T(s, a, s') [C(s, a, s') + V_n(s')]$
VI \rightarrow Asynchronous VI

• Is backing up \textit{all} states in an iteration essential?
  – No!

• States may be backed up
  – as many times
  – in any order

• If no state gets starved
  – convergence properties still hold!!
Residual wrt Value Function $V$ ($Res^V$)

- Residual at $s$ with respect to $V$
  - magnitude($\Delta V(s)$) after one Bellman backup at $s$

$$Res^V(s) = \left| V(s) - \min_{a \in A} \sum_{s' \in S} T(s, a, s') [C(s, a, s') + V(s')] \right|$$

- Residual wrt respect to $V$
  - max residual
  - $Res^V = \max_s (Res^V(s))$ 

$Res^V < \epsilon$ (\epsilon-consistency)
(General) Asynchronous VI

1. initialize $V$ arbitrarily for each state
2. while $Res^V > \epsilon$ do
3.   find a state $s$
4.   revise $V(s)$ using a Bellman backup at $s$
5.   update $Res^V(s)$
6. end
7. return greedy policy $\pi^V$
Asynch VI: Lots of Extensions to VI

• **Prioritized Sweeping**
  – select s that is likely to have the most change in V

• **Backward VI**
  – backup states in reverse order starting from goal

• **Partitioned VI**
  – divide states into partitions
  – backup partitions in reverse order from goal partition
Heuristic Search Algorithms

• Definitions

• Find & Revise Scheme.

• LAO* and Extensions

• RTDP and Extensions

• Other uses of Heuristics/Bounds

• Heuristic Design
Limitations of VI/Extensions

• **Scalability**
  – Memory linear in size of state space
  – Time at least polynomial or more

• **Polynomial is good, no?**
  – state spaces are usually huge.
  – if $n$ state vars then $2^n$ states!

• **Curse of Dimensionality!**
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_{s0}

- **S**: A set of states
- **A**: A set of actions
- **T(s,a,s')**: transition model
- **C(s,a,s')**: cost
- **G**: set of goals
- **s_0**: start state

**Under two conditions:**
- There is a *proper policy* (reaches a goal with P=1 from all states)
- Every *improper policy* incurs a cost of ∞ from every state from which it does not reach the goal with P=1
Model

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

• **What is the solution to an** $\text{SSP}_{s_0}$

• **Policy** $(S \rightarrow A)$?
  
  – are states that are not reachable from $s_0$ relevant?
  
  – states that are never visited (even though reachable)?
Partial Policy

• Define *Partial policy*
  – $\pi: S' \rightarrow A$, where $S' \subseteq S$

• Define *Partial policy closed w.r.t. a state $s$.*
  – is a partial policy $\pi_s$
  – defined for all states $s'$ reachable by $\pi_s$ starting from $s$
Partial policy closed wrt $s_0$
Partial policy closed wrt $s_0$

Is this policy closed wrt $s_0$?

- $\pi_{s_0}(s_0) = a_1$
- $\pi_{s_0}(s_1) = a_2$
- $\pi_{s_0}(s_2) = a_1$
Partial policy closed wrt $s_0$

Is this policy closed wrt $s_0$?

\[ \pi_{s_0}(s_0) = a_1 \]
\[ \pi_{s_0}(s_1) = a_2 \]
\[ \pi_{s_0}(s_2) = a_1 \]
Partial policy closed wrt $s_0$

Is this policy closed wrt $s_0$?

$\pi_{s_0}(s_0) = a_1$
$\pi_{s_0}(s_1) = a_2$
$\pi_{s_0}(s_2) = a_1$
$\pi_{s_0}(s_6) = a_1$
Policy Graph of $\pi_{s0}$

$\pi_{s0}(s_0) = a_1$
$\pi_{s0}(s_1) = a_2$
$\pi_{s0}(s_2) = a_1$
$\pi_{s0}(s_6) = a_1$
Greedy Policy Graph

- Define **greedy policy**: \( \pi^V = \arg\min_a Q^V(s,a) \)

- Define **greedy partial policy rooted at** \( s_0 \)
  - Partial policy rooted at \( s_0 \)
  - Greedy policy
  - denoted by \( \pi^V_{s_0} \)

- Define **greedy policy graph**
  - Policy graph of \( \pi^V_{s_0} \): denoted by \( G^V_{s_0} \)
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
  – $h(s) \leq V^*(s)$
Heuristic Search Algorithms

• Definitions

• Find & Revise Scheme.

• LAO* and Extensions

• RTDP and Extensions

• Other uses of Heuristics/Bounds

• Heuristic Design
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:
  – repeat
    • find a state that satisfies the two properties above
    • perform a Bellman backup
  – until no such state remains
FIND & REVISE [Bonet&Geffner 03a]

1. Start with a heuristic value function $V \leftarrow h$
2. while $V$’s greedy graph $G^V_{s_0}$ contains a state $s$ with $Res^V(s) > \epsilon$ do
3.     FIND a state $s$ in $G^V_{s_0}$ with $Res^V(s) > \epsilon$
4.     REVISE $V(s)$ using a Bellman backup at $s$
5. end
6. return a $\pi^V$

- Convergence to $V^*$ is guaranteed
  - if heuristic function is admissible
  - $\sim$no state gets starved in $\infty$ FIND steps
Heuristic Search Algorithms

• Definitions

• Find & Revise Scheme.

• LAO* and Extensions

• RTDP and Extensions

• Other uses of Heuristics/Bounds

• Heuristic Design
LAO* family

add $s_0$ to the fringe and to greedy policy graph

repeat
  - FIND: expand some states on the fringe (in greedy graph)
  - initialize all new states by their heuristic value
  - choose a subset of affected states
  - perform some REVISE computations on this subset
  - recompute the greedy graph

until greedy graph has no fringe & residuals in greedy graph small

output the greedy graph as the final policy
add $s_0$ to the fringe and to greedy policy graph

repeat
  ▪ 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
until greedy graph has no fringe & residuals in greedy graph small

output the greedy graph as the final policy
add $s_0$ in the fringe and in greedy graph

$V(s_0) = h(s_0)$
FIND: expand some states on the fringe (in greedy graph)

\[ V(s_0) = h(s_0) \]
FIND: expand some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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 some states 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
LAO*

$s_4$ was never expanded
$s_8$ was never touched
add $s_0$ to the fringe and to greedy policy graph

repeat

- 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

until greedy graph has no fringe

output the greedy graph as the final policy
Optimizations in LAO*

add $s_0$ to the fringe and to greedy policy graph

repeat

- 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 $s$
- VI iterations until greedy graph changes (or low residuals)
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy
add $s_0$ to the fringe and to greedy policy graph

repeat

- **FIND:** expand all states in greedy fringe
- initialize all new states by their heuristic value
- subset = all states in expanded graph that can reach $s$
- VI iterations until greedy graph changes (or low residuals)
- recompute the greedy graph

until greedy graph has no fringe

output the greedy graph as the final policy
iLAO* [Hansen&Zilberstein 01]

add $s_0$ to the fringe and to greedy policy graph

repeat
  - FIND: expand all states in greedy fringe
  - initialize all new states by their heuristic value
  - subset = all states in expanded graph that can reach $s$
  - only one backup per state in greedy graph
  - recompute the greedy graph
until greedy graph has no fringe

output the greedy graph as the final policy
Reverse LAO* [Dai&Goldsmith 06]

• LAO* may spend huge time until a goal is found
  – guided only by $s_0$ and heuristic

• LAO* in the reverse graph
  – guided only by goal and heuristic

• Properties
  – Works when 1 or handful of goal states
  – May help in domains with small fan in
Bidirectional LAO* [Dai&Goldsmith 06]

- Go in both directions from start state and goal
- Stop when a bridge is found
All algorithms able to make effective use of reachability information!
AO* for Acyclic MDPs [Nilsson 71]

add $s_0$ to the fringe and to greedy policy graph

repeat
  - 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 $s$
  - a single backup pass from fringe states to start state
  - recomputethe greedy graph
until greedy graph has no fringe

output the greedy graph as the final policy
Heuristic Search Algorithms

• Definitions
• Find & Revise Scheme.
• LAO* and Extensions
• RTDP and Extensions
• Other uses of Heuristics/Bounds
• Heuristic Design
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 \( \#\text{trials} \to \infty \)
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
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
start at start state
repeat
perform a Bellman backup
simulate greedy action
until hit the goal
Trial

Backup all states on trajectory

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 → ∞
RTDP Family of Algorithms

repeat
  s ← s₀

repeat //trials
  REVISE s; identify aₙₐ₉𝚎ₑ𝘥
  FIND: pick s’ s.t. T(s, aₙₐ₉ₑₑᵈ, s’) > 0
  s ← s’
until s ∈ G

until termination test
F&R and Monotonicity

- $V_k \leq_p V^* \Rightarrow V_n \leq_p V^* \ (V_n \text{ monotonic from below})$
  - If $h$ is admissible: $V_0 = h(s) \leq_p V^*$
  - $\Rightarrow V_n \leq_p V^* \ (\forall n)$

$Q^*(s,a_1) = 5 \quad Q(s, a_2) = 10$

All values $= V^*, Q^*$

$Q^*(s,a_1) < Q(s,a_2) < Q^*(s,a_2)$

$a_2$ can’t be optimal
Termination Test Take 1: Labeling

- 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

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

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

label $s$ as solved
Labeling (contd)

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

label s as solved

\[ \text{Res}^V(s) < \epsilon \]
\[ \text{Res}^V(s') < \epsilon \]

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

label s, s’ as solved
repeat
    s ← s₀
    label all goal states as solved

repeat //trials
    REVISE s; identify aₐₐₜₚₑₘₑₜ
    FIND: sample s’ from T(s, aₐₜₚₑₘₑₜ, s’)
    s ← s’
until s is solved

for all states s in the trial
    try to label s as solved
until s₀ is solved
LRTDP

• terminates in finite time
  – due to labeling procedure

• anytime
  – focuses attention on more probable states

• fast convergence
  – focuses attention on unconverged states
Picking a Successor Take 2

• Labeled RTDP/RTDP: sample $s' \propto T(s, a_{\text{greedy}}, s')$
  – Adv: more probable states are explored first
  – Labeling Adv: no time wasted on converged states
  – Disadv: labeling is a hard constraint
  – Disadv: sampling ignores “amount” of convergence

• If we knew how much $V(s)$ is expected to change?
  – sample $s' \propto \text{expected change}$
Upper Bounds in SSPs

• RTDP/LAO* maintain lower bounds
  – call it $V_l$

• Additionally associate upper bound with $s$
  – $V_u(s) \geq V^*(s)$

• Define gap(s) = $V_u(s) - V_l(s)$
  – low gap(s): more converged a state
  – high gap(s): more expected change in its value
Backups on Bounds

• Recall monotonicity

• Backups on lower bound
  – continue to be lower bounds

• Backups on upper bound
  – continues to be upper bounds

• Intuitively
  – $V_l$ will increase to converge to $V^*$
  – $V_u$ will decrease to converge to $V^*$
Bounded RTDP [McMahan et al 05]

repeat
    \[ s \leftarrow s_0 \]
    repeat //trials
        identify a_{greedy} based on \( V_l \)
        FIND: sample \( s' \propto T(s, a_{greedy}, s').\text{gap}(s') \)
        \( s \leftarrow s' \)
    until \( \text{gap}(s) < \epsilon \)

for all states \( s \) in trial in reverse order
    REVISE \( s \)

until \( \text{gap}(s_0) < \epsilon \)
If $Q_l(s,a_1) > Q_u(s,a_2)$ then $a_1$ cannot be optimal for $s$. Leads to VPI-RTDP [Sanner et al 09]
Heuristic Search Algorithms

• Definitions

• Find & Revise Scheme.

• LAO* and Extensions

• RTDP and Extensions

• Other uses of Heuristics/Bounds

• Heuristic Design
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\textsuperscript{[Dai, Mausam, Weld 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
Heuristic Search Algorithms

• Definitions

• Find & Revise Scheme.

• LAO* and Extensions

• RTDP and Extensions

• Other uses of Heuristics/Bounds

• Heuristic Design
Admissible Heuristics

- **Basic idea**
  - Relax probabilistic domain to deterministic domain
  - Use heuristics (classical planning)

- **All-outcome Determinization**
  - For each outcome create a different action

- **Admissible Heuristics**
  - Cheapest cost solution for determinized domain
  - Classical heuristics over determinized domain
Summary of Heuristic Search

- Definitions

- Find & Revise Scheme
  - General scheme for heuristic search

- LAO* and Extensions
  - LAO*, iLAO*, RLAO*, BLAO*

- RTDP and Extensions
  - RTDP, LRTDP, BRTDP, FRTDP, VPI-RTDP

- Other uses of Heuristics/Bounds
  - Action Elimination, FTVI

- Heuristic Design
  - Determinization-based heuristics
Shameless Plug
Determinization-based Algorithms for SSPs and Beyond

Mausam

Computer Science and Engineering

Indian Institute of Technology (IIT) Delhi
PPDDL

• **PDDL/STRIPS**
  – precondition: conjunction of fluents
  – effect: all changes in the state

• **PPDDL**
  – precondition: as above
  – LIST of (effect, probability)

• **RDDL**
  – concurrent effects, natural dynamics
BEYOND SSPs
Domains with Dead-ends

• Dead-end state
  – a state from which goal is unreachable

• Common in real-world
  – rover
  – traffic
  – exploding blocksworld!

• SSP/SSP$_{s0}$ do not model such domains
  – assumption of “at-least one proper policy” violated
Two Types of Dead-ends

• Explicit Dead-end

• Implicit Dead-end
Modeling Dead-ends

• How should we model dead-end states?
  – $V(s)$ is undefined for dead-ends
    ⇒ $V_I$ does not converge!!

• Proposal 1
  – Add a penalty of reaching the dead-end state $= P$

• Is everything well-formed?

• Are there any issues with the model?
Simple Dead-end Penalty $\mathcal{P}$

- $V^*(s) = \epsilon(\mathcal{P}+1) + \epsilon.0 + (1-\epsilon).\mathcal{P}$
- $= \mathcal{P} + \epsilon$
- $V(\text{non-deadend}) > \mathcal{P}$
Proposal 2

- fSSPDE: Finite-Penalty SSP with Deadends
- Agent allowed to stop at any state
  - by paying a price = penalty $\mathcal{P}$
  - Intuition: achieving a goal is worth $-\mathcal{P}$ to the agent

$$V^*(s) = \min \left( \mathcal{P}, \min_{a \in \mathcal{A}} \sum_{s' \in S} \mathcal{T}(s, a, s')\mathcal{C}(s, a, s') + V^*(s') \right)$$

- Equivalent to SSP with special $a_{\text{stop}}$ action
  - applicable in each state
  - leads directly to goal by paying cost $\mathcal{P}$
- SSP = fSSPDE
fSSPDE Algorithms

• All SSP algorithms applicable...

• Efficiency: unknown so far...
  – Efficiency hit due to presence of deadends
  – Efficiency hit due to magnitude of $\mathcal{P}$
  – Efficiency hit due to change of topology (e.g., TVI)
SSPADE: Dead Ends are Avoidable from $s_0$

[Kolobov, Mausam, Weld, UAI’12]

- D.e.s may be avoidable *from* $s_0$ via an optimal policy

- Can’t compute $V^*(s)$ for every state
- But need only “relevant” states to get the “right” value
- there exists a proper (partial) policy rooted at $s_0$
SSPADE

• Can be solved with optimal heuristic search from $s_0$
  – FIND shouldn’t starve states; REVISE should halt

• Heuristic Search Algorithms
  – LAO*: may not converge
    • $V$(dead-ends) will get unbounded: VI may not converge
  – iLAO*: will converge
    • only 1 backup $\Rightarrow$ greedy policy will exit dead-ends
  – RTDP/LRTDP: may not converge
    • once stuck in dead-end $\Rightarrow$ won’t reach the goal
    • add max #steps in a trial… how many? adaptive?
Unavoidable Dead-ends

• fSSPUDE: Finite-Penalty SSP with Unavoidable Dead-Ends [Kolobov et al 12]
  – same as fSSPDE but now with a start state

• Same transformation applies
  – add an $a_{\text{stop}}$ action from every state

• $\text{SSP}_{s_0} = \text{fSSPUDE}$
Unavoidable Dead-ends

- iSSPUDE: Infinite-Penalty SSP with Unavoidable Dead-Ends [Kolobov et al 12]
  - (MAXPROB) find policy that first maximizes the prob of reaching goal
  - from all such policies find one minimum expected cost

- Dual objective

- iSSPUDE is much harder than SSP$_{s0}$
DETERMINIZATION-BASED APPROXIMATION ALGORITHMS
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
Approximation Ideas

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
  – Others possible, e.g., optimal in the limit vs. suboptimal in the limit

• Most techniques assume factored fSSPUDE MDPs (SSP$_{s0}$ MDPs with a finite dead-end penalty)

• Approaches differ in the quality aspect they sacrifice
  – Probability of reaching the goal
  – Expected cost of reaching the goal
  – Both
Example Domain (cont’d)
\( \text{SSP}_{s_0} \text{ MDP} \)

- \( S \): A set of states
- \( A \): A set of actions
- \( T(s,a,s') \): transition model
- \( C(s,a,s') \): action cost
- \( s_0 \): start state
- \( G \): set of goals

GetW, GetH, GetS, Tweak, Smash
Outline

• Online Algorithms
  – FF Replan
  – FF Hindsight
  – RFF

• Offline Algorithms
  – Inadmissible Heuristics
  – Dimensionality Reduction
  – Other Determinization Approaches
Online Algorithms: Motivation

- **Defining characteristics:**
  - Planning + execution are interleaved
  - Little time to plan
    - Need to be fast!
  - Worthwhile to compute policy only for visited states
    - Would be wasteful for all states
Determinization-based Techniques

- A way to get a quick’n’dirty solution:
  - Turn the MDP into a *classical* planning problem
  - Classical planners are comparatively very fast

- Main 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
All-Outcome Determinization

Each outcome of each probabilistic action $\rightarrow$ separate action

$P = \frac{9}{10}$

$P = \frac{1}{10}$
Most-Likely-Outcome Determinization

P = 4/10

P = 6/10

G e t H

X

X

X
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: Details

• Uses either the AO or the MLO determinization
  – MLO is smaller/easier to solve, but
  – AO contains all possible plans, but

• Uses the *FF* planner to solve the determinization
  – Super fast
  – Other fast planners, e.g., LAMA, possible

• Does not cache computed plans
  – Recomputes the plan in the 3rd step in the example
FF-Replan: Theoretical Properties

• Optimizes the *MAXPROB* criterion in SSPs
  – In SSPs, this is always 1.0
  – Super-efficient on SSPs w/o dead ends
  – Largely ignores expected cost

• 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-Replan and Dead Ends

Deterministic plan:

Its possible execution:
Putting “Probabilistic” Back Into Planning

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

• **Sample determinizations probabilistically!**
  – Hopefully, probabilistically unlikely plans will be rarely found

• Basic idea behind **FF-Hindsight**
FF-Hindsight: Overview

(Estimating Q-Value, Q(s,a))

S: Current State, A(S) → S’

1. For Each Action A, Draw Future Samples

Each Sample is a Deterministic Planning Problem

2. Solve Time-Dependent Classical Problems

See if you have goal-reaching solutions, estimate Q(s,A)

3. Aggregate the solutions for each action

\[ \text{Max}_A Q(s,A) \]

4. Select the action with best aggregation

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
FF-Hindsight: Example

Objective: Optimize MAXPROB criterion

Left Outcomes are more likely

Action

Probabilistic Outcome

Time 1

Time 2

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

Maximize Goal Achievement

Time 1

A1
A2

Time 2

A1
A2

A1: 1
A2: 0

Dead End
Goal State

Action
Probabilistic Outcome

Left Outcomes are more likely

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati
FF-Hindsight: Sampling a Future-3

Maximize Goal Achievement

A1

A2

A1

A2

A1

A2

A1

A2

Left Outcomes are more likely

Time 1

Time 2

Action

Probabilistic Outcome

A1: 2

A2: 1

Action

State

Dead End

Goal State
FF-Hindsight: Details & Theoretical Properties

• For each s, FF-Hindsight samples \( w \) \( \textit{L-horizon futures} \) \( F^L \)
  – In factored MDPs, amounts to choosing \( a \)'s outcome for each \( h \)

• Futures are solved by the FF planner
  – Fast, since they are much smaller than the AO determinization

• With enough futures, will find MAXPROB-optimal policy
  – If horizon \( H \) is large enough and a few other assumptions

• Much better than FF-Replan on MDPs with dead ends
  – But also slower – lots of FF invocations!
Providing Solution Guarantees

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

• FF-Hindsight provides some theoretical guarantees
  – Practical implementations distinct from theory
  – Wastes solutions: generates them, then forgets them

• RFF (Robust FF) provides quality guarantees in practice
  – Constructs a policy tree out of deterministic plans
RFF: Overview
[Teichteil-Königsbuch, Kuter, Infantes, 2010]

Make sure the probability of ending up in an unknown state is $< \varepsilon$
1. Generate either the AO or MLO determinization. Start with the policy graph consisting of the initial state $s_0$ and all goal states $G$. 
2. Run FF on the chosen determinization and add all the states along the found plan to the policy graph.
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.
RFF: Run VI (Optional)

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. $s_0$.

$P(\text{failure}) = ?$

If $P(\text{failure}) > \varepsilon$

Else, done!
RFF: Finding Plans from the Fringe

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
RFF: Theoretical Properties

- Fast
  - FF-Replan forgets computed policies
  - RFF essentially memorizes them

- When using AO determinization, guaranteed to find a policy that with $P = 1 - \varepsilon$ will not need replanning
Summary of Determinization Approaches

• Revolutionized SSP MDPs approximation techniques
  – Harnessed the speed of classical planners
  – Eventually, “learned” 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 the expected 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
Outline

• **Online Algorithms**
  – FF Replan
  – FF Hindsight
  – RFF

• **Offline Algorithms**
  – Inadmissible Heuristics
  – Dimensionality Reduction
  – Other Determinization Approaches
Moving on to Approximate Offline Planning

• Useful when there is no time to plan as you go ...
  – E.g., when playing a fast-paced game
Inadmissible Heuristic Search

• Why?
  – May require less space than admissible heuristic search
The FF Heuristic

- Taken directly from deterministic planning
  - A major component of the formidable FF planner

- Uses the all-outcome determinization of a PPDDL MDP
  - But ignores the delete effects (negative literals in action outcomes)
  - Actions never “unachieve” literals, always make progress to goal

- \( h_{FF}(s) = \text{approximate cost of a plan from } s \text{ to a goal in the delete relaxation} \)

- Very fast due to using the delete relaxation

- Very informative

[Hoffmann and Nebel, 2001]
The GOTH Heuristic

• 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

• Uses AO determinization and the FF planner

[Kolobov, Mausam, Weld, 2010]
GOTH Overview

1. Start running an MDP solver (e.g., LRTDP)
2. Analyze the MDP (MDP M)
3. Evaluate the state (s)
4. Plan precondition & cost
5. Regress plan
6. Evaluate state (s)
7. Policy

GOTH

- Determinize M
- AOdet(M)
- Run a classical planner (e.g., FF)
- Plan
- Plan precondition & cost
- Evaluate state (s)
- hGOTH (s)
Regressing Trajectories

Plan

Precondition costs

Precondition costs

= 1

= 2
Plan Preconditions
Estimating State Values

• Intuition
  – Each plan precondition cost is a “candidate” heuristic value

• Define $h_{GOTH}(s)$ as MIN of all available plan precondition values applicable in $s$
  – If none applicable in $s$, run a classical planner and find some
  – Amortizes the cost of classical planning across many states
What about Dead Ends?

• How to find an implicit dead-end state?
  – FF doesn’t return a solution (say in some fixed time)

• GOTH generalizes each successful trajectory.
  – Can we generalize each implicit dead-end state?

• SixthSense!
GOTH Overview

Start running an MDP solver (e.g., LRTDP)

MDP M

Policy

GOTH

Determinize M

AOdet(M)

Run a classical planner (e.g., FF)

Plan

Regress plan

Plan precondition & cost

Evaluate s

State s

h_{GOTH}(s)
Start running an MDP solver (e.g., LRTDP)
Research Question

Can we devise a sound identification procedure fast enough to obviate memoization?

Learns feature combinations whose presence guarantees a state to be a dead end.
Generate-and-Test Procedure

[Kolobov, Mausam, Weld 2010]

• **Generate a nogood candidate**
  - Key insight: Nogood = conjunction that *defeats* all known plan preconditions
  - For each plan precondition, pick a literal that defeats it

• **Test the candidate**
  - Needed for *soundness*, since we don’t know all preconditions
  - Use the non-relaxed Planning Graph algorithm
Generating a Nogood Candidate

Compute histogram of literal occurrence in dead ends:

Training dead ends:
Generating a Nogood Candidate

Dead-end literal occurrence stats

Current basis function

Current nogood candidate
Generating a Nogood Candidate

Dead-end literal occurrence stats

Current basis function

Current nogood candidate
Testing the Candidate

Nogood

Planning graph

Literals over all vars not in nogood
Testing the Candidate

• If the Planning Graph fails to reach the goal, the candidate is a nogood.
  – Planning Graph is complete, hence this is sound

• Note: in is superfluous
  – Remove literals one-by-one and test as above to get
Scheduling

• Need to invoke learning more than once

• Never know how much training data is “enough”

• Solution: adaptive scheduler
  – Finds a “good” amount of training data, invokes learning accordingly
Benefits of SixthSense

- Can act as submodule of many planners and ID dead ends
  - By checking discovered nogoods against every state
$h_{FF}$ vs GOTH+6S

Figure 1: GOTH outperforms $h_{FF}$ on Machine Shop, Triangle Tireworld, and Blocksworld by a large margin both in speed...

Figure 2: ... and in memory
Outline

• Online Algorithms
  – FF Replan
  – FF Hindsight
  – RFF

• Offline Algorithms
  – Inadmissible Heuristics
  – Dimensionality Reduction
  – Other Determinization Approaches
• Determinization
  – Determinize the MDP
  – Classical planners \textit{fast}
  – E.g., FF-Replan
  – Cons: may be troubled by
    • Complex contingencies
    • Probabilities

• Function Approximation
  – Dimensionality reduction
  – Represent state values with basis functions
    • E.g., \( V^*(s) \approx \sum_i w_i b_i(s) \)
  – Cons:
    • Need a human to get \( b_i \)

\textbf{ReTRASE}

\textbf{Marry these paradigms to extract problem-specific structure in a fast, problem-independent way.}
ReTrASE

- Largely similar to $h_{GOTH}$
  - Uses preconditions of deterministic plan to evaluate states

- For each plan precondition $p$, defines a **basis function**
  - $B_p(s) = 1$ iff $p$ holds in $s$, $\infty$ otherwise

- Represents $V(s) = \min_p w_p B_p(s)$
  - Thus, the parameters are $w_p$ for each basis function
  - Problem boils down to learning $w_p$
  - Does this with modified RTDP

- **Crucial observation:** # plan preconditions sufficient for representing $V$ is typically much smaller than $|S|$}
  - Because one plan precondition can hold in several states
  - Hence, the problem dimension is reduced!

[Kolobov, Mausam, Weld, 2009]
Exploding Blocks World: Success Rate

% of Successful Trials

Exploding Blocks World’06 Problem #
ReTrASE Theoretical Properties

• Empirically, gives a large reduction in memory vs LRTDP

• Produces good policies (in terms of MAXPROB) when/if converges

• Not guaranteed to converge (weights may oscillate)

• No convergence detection/stopping criterion
Current Trend: Deep Probabilistic Planning

• Use deep RL ideas
  – for PPDDL or RDDL planning

• Ideas
  – use given model effectively
  – transfer between problem instances
Outline

• Online Algorithms
  – FF Replan
  – FF Hindsight
  – RFF

• Offline Algorithms
  – Inadmissible Heuristics
  – Dimensionality Reduction
  – Other Determinization Approaches
Self-Loop Determinization

\[ T = 0.9 \]
\[ C = 1 \]

\[ T = 1.0 \]
\[ C = \frac{1}{0.9} = 1.11 \]

\[ T = 1.0 \]
\[ C = \frac{1}{0.1} = 10 \]
Self-Loop Determinization

• Like AO determinization, but modifies action costs

• “Unlikely” deterministic plans look expensive in SL det.!

• Used in the HMDPP planner
Space of Determinizations

[Pineda & Zilberstein, 2017]

• Extreme 1
  – most likely outcome determinization

• Extreme 2
  – all outcome determinization

• Middle ground
  – primary outcome (upto l) determinization

• Extreme 1
  – all actions deterministic

• Extreme 2
  – all actions completely probabilistic

• Middle ground
  – actions have primary outcomes + (upto k) exception outcomes
BEYOND SSPs (contd)
Unavoidable Dead-ends (contd)

• iSSPUDE: Infinite-Penalty SSP with Unavoidable Dead-Ends [Kolobov et al 12]
  – (MAXPROB) find policy that first maximizes the prob of reaching goal
  – from all such policies find one minimum expected cost

• Dual objective

• iSSPUDE is much harder than SSP_{s0}
MAXPROB: Dealing with Unavoidable Infinitely Damaging Dead Ends

- Comparing policies in terms of cost meaningless
- MAXPROB/GSSP MDPs: evaluate policies by probability of reaching goal
  - Set all action costs to 0 (they don’t matter), reward 1 for reaching goal
  - Fixed-point methods such as VI or LRTDP don’t converge because of traps

[Kolobov, Mausam, Weld, Geffner ICAPS’11]
(Maximization) MDP Examples

SSP

GSSP

S3P
Generalized SSPs: Example

Solution

Not a solution
GSSPs: Is $V^*$ A Fixed Point of $B$?

• Reminder: in SSPs, $V^* = B \, V^*$, where
  – $B$ is the *Bellman backup operator*
  
  – $B \, V(s) = \max_a \{ R(s, a) + \sum_{s' \in \text{succ}(s,a)} T(s, a, s') \, V(s') \}$

• In SSPs, $V^*$ is a fixed point of $B$
  – *Still true in GSSPs:*

![Diagram of a graph with nodes labeled with values and edges showing transitions between states.](attachment:image.png)
GSSPs: Is \( V^* \) The **Unique** Fixed Point of \( B \)?

- In SSPs, \( V^* \) is the unique fixed point of \( B \)
  - I.e., \( V^* = B \circ B \circ \ldots B \circ V_0 \), \( V_0 \) is a heuristic value function
  - Not true in GSSPs:

- Moreover, all suboptimal fixed points are admissible!
GSSPs: Is Every $V^*$-greedy $\pi$ A Solution?

- In SSPs, every $\pi$ greedy w.r.t $V^*$ reaches the goal
  - Not true in GSSPs:
Efficiently Solving GSSPs: Attempt #1

• **Just Run F&R!**
  
  – Start with an admissible $V_0$

  ![Diagram](image)

  – Done!
Attempt #1: What Went Wrong?

- In GSSPs, suboptimal fixed points are admissible!
  - When starting with $V_0 \geq V^*$, F&R hit one of them.
  - $B$ can’t change $V$ over *traps* – strongly connected components in $V$’s greedy graph

- Can yield an arbitrarily poor solution
Efficiently Solving GSSPs: FRET

• **Find, Revise, Eliminate Traps**
  – First heuristic search algorithm for MDPs beyond SSP
  – Provably optimal if the heuristic is admissible

• **Main idea**
  – Run F&R until convergence
  – Eliminate traps in the policy envelope
  – Repeat until no more traps
FRET Example: Finding $V^*$

Start with an admissible $V_0$

Run F&R until convergence

Eliminate Traps in the resulting $V_i$

Find-and-Revise

Eliminate Traps

Find-and-Revise

No traps left – done!
FRET Example: Extracting $\pi^*$

- Iteratively “connect” states to the goals
  - Using optimal actions
  - Until $s_0$ is connected
Goal-Oriented MDP Hierarchy
Thanks!