Summary of part II

Basic Methods
- No Structure assumed or used
- Planning: VI, PI, MPI, LinProg, RTDP
- Learning/Opt: TD/SARSA, Q-Learning
- Learn/Opt/Plan: Dyna approach

Better Methods
- Still: No Structure assumed or used
- Improved algorithms: LAO*, L-RTDP
- Improve dynamic programming to focus on reachable states + improve convergence properties
- "Polynomial guarantees" for Learn/Opt algorithms: E^3, RMAX
- Rely on distinctions of Known/unknown states or entries + optimal planner
- [More results: Delayed-Q & extensions]

State Aggregation
- How do we handle large state spaces?
- Arbitrary partitions
- Feature based representation: $\phi(s, a)$
- Approach I: Logical partitions of space
- Approach II: Linear function approximation
  $Q(s, a) = w^T \cdot \phi(s, a)$
- Or we may need to refer to and use objects and relations to describe states and partitions
## Factored Representations (1)
- Structure is given by representing state via features (basis functions)
- Planning, Logic Based Dynamic Prog: SPUD, APPRICODD
- State partitions are dynamically and automatically calculated by being routed to the same leaf of ADD

## Factored Representations (2)
- [we did not cover this]
- Improved algorithms:
  - Factored versions of LAO*, RTDP, ...
  - Factored versions of $E^3$, ...
- (dynamically derived) Logical partitions of states

## Factored Representations (3)
- Structure is given by representing state via features (basis functions)
- Planning, linear function approximation: linear-API, linear-ALP
- Seek “best” linear approximation for true value function in P-eval or LP formulation

## Factored Representations (4)
- Structure is given by representing state via features (basis functions)
- Planning, linear function approximation: LSP
  \[ Q(s, a) = w^T \cdot \phi(s, a) \]
- Seek “best” linear approximation for true value function in P-eval or LP formulation
- Does not need a model!

## Relational Representations (1)
- Structure is given by representing state via objects and relations
- Planning, Logic Based: SDP, ReBel, FODD-Planner
- State partitions are dynamically and automatically calculated by being routed to the same leaf of FODD

## Relational Representations (2)
- [we did not cover this]
- Structure is given by representing state via objects and relations
- Define features over this space and
  \[ Q(s, a) = w^T \cdot \phi(s, a) \]
- Planning, linear function approximation:
  Relation linear-API, Rel-ALP
- Seek “best” linear approximation for true value function in P-eval or LP
### Relational Representations (3)
- Structure is given by representing state via objects and relations
- Relational RL
- Monte Carlo sampling
- Learn explicit representation of Q function (and/or policy) as a relational decision tree
- State partitions automatically calculated
- [many extensions/variations not covered]

### Relational Representations (4)
- Structure is given by representing state via objects and relations
- Non-parametric policy gradients (first done for factored; this version applies for all)
- Represent Q as a sum of “smaller” Q f’s
- Each is (a learned version) of the gradient of the quality of policy based on previous Q

### State Aggregation: recap
- How do we handle large state spaces?
- Arbitrary partitions
- Logical partitions of space
- Linear function approximation
  - Both calculate complete solutions
- + more algorithms: policy gradients

### Other Algorithmic Approaches
- How do we handle large state spaces?
- Arbitrary partitions
- Logical partitions of space
- Linear function approximation
  - Both calculate complete solutions
- + more algorithms: policy gradients
- Sampling: calculate “best action” for current state and continue to next

### Sampling/Search
- Sampling: calculate “best action” for current state and continue to next
- Policy improvement via Policy rollout
- Sparse sampling: optimal
- UCT: adaptive sparse sampling: optimal

### Other Algorithmic Approaches
- Logical partitions of space
- Linear function approximation
- + more algorithms: policy gradients
- Sampling: calculate “best action” for current state and continue to next
- By solving the corresponding probabilistic inference problem
Planning via Inference in DBNs

- Planning as inference for parameters of policy
- EM formulation; Applied to simple problems
- Factored/Relational problems more complex
- Use simple forward sampling

More Complex Problems

- We focused on Large state spaces
- Multiple/parallel actions: imply huge action space
- Exogenous Events: imply huge branching to next states even for single action
- Action durations: problem may not be Markov w/o state extension

More Complex Problems

- Engineering solution: identify useful features and use linear function approximation

Multiple/parallel actions

- Action selection phase: pick best tuple of actions
- Learn/Plan phase: calculate Q or policy over tuples of actions
- Ideas: prune obvious low-value combinations; sample some combinations; use greedy/hill-climb search

Exogenous Events

- Even for one action, the number of possible next states is huge
- Ideas:
  - Some methods (TD, SPUDD related) not directly affected by branching factor (although result might)
  - Use "after state" representation and sampling

Action Durations

- Problem not Markov
- Ideas:
  - Can extend state to include info re actions and remaining durations
  - But must deal with huge state space and huge action space
### Action Durations

- GSMDP formulation: transition encodes extra info; non Markov
- Translate/approx by continuous time Markov
- Translate/approx by discrete time Markov
- Use solution in original problem
- But must deal with (huge state space and) huge action space

---

### A few more things

- On web page but we did not cover

---

### Hierarchical Representations

- Structure is captured by a hierarchy of problems and policies ("Taxi" problem)
- User provides Hierarchy or partial program and algorithm learns parameters:
  - MAXQ and ALISP approaches
- Parameters can be learned
- How are factored and hierarchical approaches related?

---

### Learning from Demonstration: L2Act

- Observe "Expert" (or optimal policy) acting in the domain
- Example are \((s,a)\) pairs
- Directly learn a representation of the policy
- Theoretically justified
- Applied to large planning problems

---

### Learning variant of API

- Policy improvement step approx by:
  - Generating examples for Q of improved policy via rollout
  - Directly learn a representation of the improved policy
- Applied to large planning problems

---

### Inverse RL

- Observe "Expert" (or optimal policy) acting in the domain
- Example are states visited
- Assume transition model known but reward is not
- "Learn reward" and then plan using any algorithm
- Nice results with linear approximation; sufficient to approx feature expectations
- Impressive demonstrations
Reward Shaping

- Given MDP
- We can change the reward function
- Such that optimal solution does not change
- But learning can happen much faster

Model Learning in RL

- Learning transition model and reward and plan (RMAX, E^3, Dyna, ...)
- Tricky for relational worlds
- Some heuristics exist
- Some formal results

Partially Observable MDPs

- The state is not fully observable during execution
- Agent must capture "belief state" re where they are
- And choose actions accordingly
- Much more difficult computationally

“Natural” Representations

- Predictive State Representations (also addresses partial observability)
  Represent states using their predictions on results of tests.
  Discover "just the right" distinctions of history and state for optimal control

“Natural” Representations

- Proto-Value functions and Rep-PI
- Discover "the right" feature representation (manifold) for a problem by deriving it from the geometry of transitions/value in the domain
- Then learning/planning is easier
- Value function is smooth over representation.

Summary

- Good set of basic techniques
- Multiple approaches for factored and relational problems
- Some work on more complex problems
- Multiple novel algorithmic ideas
- Multiple alternative ways to think about (the right) representation