Planning, Execution & Learning: Reactive Planning

Reid Simmons
The Problem With Policies

- Very Expensive to *Generate*

- Very Expensive to *Store*

- May be Expensive to *Access*
  - Markov policies (linear in size of state space)
  - Universal and Teleo-Reactive plans (logarithmic)
  - RAPs (bounded)
  - Real-time search (bounded)
Policy Issues

- How to Represent Policies for Efficient Retrieval

- Which Plans/Policies to Cache

- When to Plan and When to Execute/React

- How to Avoid Having Sensing Become the Bottleneck
  - Limited sensors
  - Partially observable environment

- How to Detect/Handle Cyclic Behavior
Universal Plans (Schoppers 1987)

- Complete Mapping From Sensors to *Conditions*
  - Can take duration of actions into account
  - Can take advantage of dynamics of environment
  - Influenced by PRS (Georgeoff), REX (Kaelbling), and robotics (control theory)
- Implements Policy as *Decision Tree*
  - Answers "what to do next"
    - Sequencing encoded in structure of decision tree
    - No notion of *error*
  - Treats planning & plan selection as classification problem
- Can be Synthesized Automatically
  - Uses back-chaining, non-linear planner
  - Break into action-sized chunks, with appropriate sensing actions
**Universal Block-Stacking Plan**

```
| on(a, b)? |
|---|---|
| T | clear(b)? |
| F | holding(a) |

```

1. **top?**
   - T: no-op
   - F: ~holding(a)
     - T: raise
     - F: open
2. **clear(b)?**
   - T: holding(a)
   - F: over(b)?
     - T: lower
     - F: top?
       - T: lateral
       - F: raise

---

Planning, Execution & Learning: Reactive

Simmons, Veloso : Fall 2001
Dealing With the State Explosion

• Decision Trees Make Classification More Efficient
  – Proportional to number of features (although tree itself is exponential)

• Use General (Variablized) Rules

• Use Efficient State Representations (e.g. BDDs)

• Use at Multiple Levels of Abstraction
  – Coarse-grained and fine-grained universal plans

Ultimately, in Most Cases, Need to Choose What to Plan For…
**Entropy Reduction Engine (ERE)**
*(Drummond & Bresina, 1990)*

- Overall Architecture for Generating and Executing Reactive Plans
  - Incrementally compiled from domain models

---

**Problem Reduction** → **Strategies** → **Temporal Projection** → **Rules** → **Execution (Reactor)** → **Actions**

- **Reactor**: Choose applicable rule and apply action
- **Projector**: Produce “plans” and compile rules
- **Reductor**: Decompose problem into subproblems

---

*Planning, Execution & Learning: Reactive*
Situated Control Rules (SCRs)

- “If-Then” Rule Describing Action to Take in Given Situation
  - Represents single step along way to achieving a goal
    - “if <situation> & <goal> then <action>”
    - “local control program”
    - Similar to CIRCA’s TAPs
  - Utilizes both sensor and internal state information
  - May not have applicable rules for all situations
    - Opportunistically created by the projector
  - Does not address the problem of choosing (arbitrating) amongst applicable rules
**ERE’s Temporal Projector**

- Probabilistic, Linear Planner
  - Handles goals of achievement, prevention, and maintenance
  - Forward projection of non-deterministic actions
    - Can handle exogenous events
    - Uses *beam search* to control projection (estimate of work remaining to achieve goal)
    - “Robustify” initial plan by adding contingency branches
      - Attend to high probability deviations
- Compilation of SCRs
  - Uses goal regression and *explanation-based learning* (EBL) to form a generalization of <state, action, goal> triples
  - “Anytime” nature ensures reactivity
Agent-Centered Search (Koenig 1997)

- Allow Bounded Amount of Search (Lookahead) to Determine Next Action to Execute
  - Incrementally update value function
    - Incrementally create optimal policy
    - Akin to reinforcement learning
  - Can trade off planning time (=> plan quality) and execution speed
  - Handles uncertainty by acting, which may gain information

- Several Theoretical Results
  - Complexity of class of agent-centered search algorithms
  - Influence domain properties can have on complexity
**Min-Max LRTA**

- Extension to Non-Deterministic Domains of Korf’s Learning Real-Time A* Algorithm
  - No probabilistic information: Assume worst case for agent (where nature is the “opponent”)
  - \( u(s) = -1 + \max_{a \in A(s)} \min_{s' \in \text{succ}(s, a)} u(s') \)
  - Can eventually learn optimal policy
- Also learns while trying to reach goal for the first time

*Observe: OWOW*

*Action: Forward or Reverse?*
Complexity Results

- Min-Max LRTA* has Tight Bounds of $O(n^2)$ Action Occurrences Over *All* Domains
  - No algorithm that performs constant lookahead can do better, over all possible domains
- Q-Learning is $O(n^3)$ if it uses “dense” reward structure
  - Penalize actions *or* initialize Q-values to non-zero
  - Otherwise can be exponential
- Undirected and Directed *Eulerian* Domains are “Easier” to Search, in General
- Domains with Small Maximum Goal Distance are “Easier” to Search, on Average