Conformant Probabilistic Planning via CSPs

ICAPS-2003
Nathanael Hyafil & Fahiem Bacchus
University of Toronto
Contributions

• Conformant Probabilistic Planning
  – Uncertainty about initial state
  – Probabilistic Actions
  – No observations during plan execution
  – Find plan with highest probability of achieving the goal

• Utilize a CSP Approach
  – Encode the problem as a CSP
  – Develop new techniques for solving this kind of CSP

• Compare Decision-Theoretic algorithms
Conformant Probabilistic Planning

**CPP Problem** consists of:

- **S**: (finite) set of states (factored representation)
- **B**: (initial) probability distribution over S (belief state)
- **A**: (finite) set of (prob.) actions (represented as sequential-effect trees)
- **G**: set of Goal states (Boolean expression over state vars)
- **n**: length/horizon of the plan to be computed

**Solution**: find a plan (finite sequence of actions) that maximizes the probability of reaching a goal state from **B**
Example Problem: SandCastle

- $S$: \{\text{moat, castle}\}, both Boolean
- $G$: $\text{castle} = \text{true}$
- $A$: sequential effects tree representation (Littman 1997)

```
Erect-Castle

\begin{array}{c|c}
\text{castle} & \text{moat} \\
\hline
T & 1.0 \\
F & R_1 \quad R_2 \\
& 0.67 \quad 0.25 \\
\hline
\text{castle} & \text{moat} \\
\hline
T & 0.75 \\
F & R_3 \\
& 1.0 \\
\hline
\end{array}
```
Constraint Satisfaction Problems

- **Encode the CPP as a CSP**
- **CSP**
  - finite set of variables each with its own finite domain
  - finite set of constraints, each over some subset of the variables.
- **Constraint**: satisfied by only some variable assignments.
- **Solution**: an assignment to all Variables that satisfies *all* Constraints
- **Standard Algorithm**: Depth-First Tree Search with Constraint Propagation
The Encoding

• Variables:
  – State variables (usually Boolean)
  – Action variables \{1, \ldots, |A|\}
  – Random variables to encode the action probabilities (True/False/Irrelevant).
    • The setting of the random variables makes the actions deterministic
    • Random variables are independent

• Constraints
  – Initial belief state represented as an “Initial action” (as in partial order planning).
  – Actions constraint the state variables at step i with those at step i+1
    • Each branch of the effects trees is represented as one constraint.
  – A constraint representing the goal
State Var
Action Var

T F
V₁
V₂
a₁ a₂ a₃

Diagram showing the state variables (V₁, V₂) and action variables (a₁, a₂, a₃).
State Var
Action Var
Random Var
So far

• Encoded CPP as a CSP
• Solved the CSP

• How do we now solve the CPP?
CSP
Solutions
CSP solution:

assignment to State/Action/Random variables that is
- valid sequence of transitions
- from initial
- to goal state
State Var
Action Var
Random Var
Goal States

Action variables: plan executed
State variables: execution path induced by plan
Probability of a Solution

Product prob of Random vars : probability that this path was traversed by this plan
Value of a Plan

Value of plan $\pi$:
$\Sigma$ probs of all solutions with plan = $\pi$

After all $\pi$’s evaluated: optimal plan = one with highest value
Redundant Computations

- Due to the Markov property if the same state is encountered again at step $i$ of the plan the subtree below will be the same
  - If we can cache all the info in this subtree: explore only once
- To compute the best overall $n$-step plan: need to know value of every $n-1$ step plan for all states at step $l$. 
Caching

- Probability of success (value) of a $i$ step plan $<a, \pi>$ in state $s = \text{expectation of } \pi$’s success probability over the states reached from $s$ by $a$:

- If we know the value of $\pi$ in each of these states: can compute its value in $s$ without further search

- So, for each state $s$ reached at step $i$, we cache the value of all $n-i$ step plans
CPplan Algorithm

- CPplan():
  Select next unassigned variable V
  If V is last state var of a step:
    If this state/step is cached return
  Else if all vars are assigned (must be at a goal state):
    Cache 1 as value of previous state/step
  Else
    For each value d of V
      V=d
      CPplan()
      If V is some Action var A^i
        Update Cache for previous state/step
        with values of plans starting with d
Caching Scheme

- Needs a lot of memory
  - proportional to $|S| \times |A|^n$
  - no known algorithm does better
- Other features
  - Cache key simple (state / step)
  - Partial Caching achieves a good space/time tradeoff
MAXPLAN (Majercik & Littman 1998)

- Parallel approach based on Stochastic SAT
- Caching Scheme different
- Faster than Buridan and other AI Planners

- uses (even) more memory than CPplan
- 2 to 3 orders of magnitude slower than CPplan
Results vs. Maxplan

**SandCastle-67**

- **MaxPlan**
- **Cpplan**

**Slippery Gripper**

- **Maxplan**
- **Cpplan**
CPP as special case of POMDPs

- POMDP: model for probabilistic planning in partially observable environments
- CPP can be cast as a POMDP in which there are no observations.
- Value Iteration, a standard POMDP algorithm, can be used to compute a solution to CPP.
Value Iteration: Intuitions

- Value Iteration utilizes a powerful form of state abstraction.
  - Value of an $i$-step plan (for every belief state) is represented compactly by vector of values (one for each state): value on a belief state is the expectation of these values.
  - This vector of values is called an $\alpha$-vector.
  - Value iteration need only consider the set of $\alpha$-vectors that are optimal for some belief state.
  - Plans optimal for some belief state are optimal over an entire region of belief states.
  - So regions of belief states are managed collectively by a single plan ($\alpha$-vector) that is $i$-step optimal for all belief states in the region.
\(\alpha\)-vector Abstraction

- Number of alpha-vectors that need to be considered might grow much more slowly than the number of action sequences.

- Slippery Gripper:
  - 1 step to go: 2 \(\alpha\)-vectors instead of 4 actions
  - 2 steps to go: 6 instead of 16 (action sequences)
  - 3 steps to go: 10 instead of 64
  - 10 steps to go: 40 instead of \(>10^6\)
Results vs. POMDP

Slippery Gripper

Grid 10x10
Dynamic Reachability

- POMDPs: small portion of all possible plans to evaluate but on all belief states including those not reachable from the initial belief state.
- Combinatorial Planners (CPplan, Maxplan) must evaluate all $|A|^n$ plans but tree search performs dynamic reachability and goal attainability analysis to only evaluate plans on reachable states at each step.
- Ex: Grid 10x10, only 4 states reachable in 1 step
Conclusion -- Future Work

- New approach to CPP, better than previous AI planning techniques (Maxplan, Buridan)
- Analysis of respective benefits of decision theoretic techniques and AI techniques

- Ways to combine abstraction with dynamic reachability for POMDPs and MDPs.
Results vs. Maxplan

SandCastle - 67

Slippery Gripper

Time — Number of Steps

Time — Number of Steps

CPU Time

Number of Steps
Results vs. POMDP

Slippery Gripper

Grid 10x10