Planning Graphs (Pre Lecture)

Dr. Neil T. Dantam

CSCI-534, Colorado School of Mines

Spring 2020

Outline

Planning Graphs
- Construction
- Analysis

Planning with Planning Graphs
- GraphPlan
- GraphPlan+SATPlan (BlackBox)

Introduction

Planning Graphs

- A graph
- Represents structure of a planning domain
- Useful as a heuristic
- Fast to construct

Outcomes

- Know the different parts of a planning graph
- Construct the planning graph for a given domain
- Use the planning graph to plan

Outline

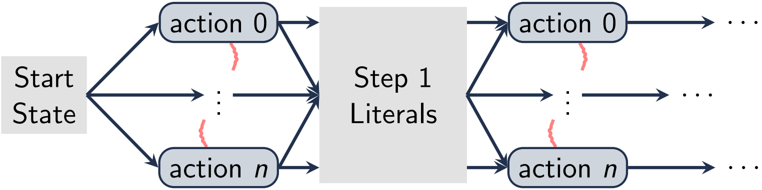
- Planning Graphs
  - Construction
  - Analysis
- Planning with Planning Graphs
  - GraphPlan
  - GraphPlan+SATPlan (BlackBox)

Planning Graph Overview

Nodes: literals $\cup$ actions

Edges:
- Transition: connects actions with precondition and effect literals, $(\ell \times a) \cup (a \times \ell)$
- Mutex: conflicts (mutual exclusion) between actions and literals, $(\ell \times \ell) \cup (a \times a)$

Levels: Sequences of levels: timesteps



Example: Cake Domain

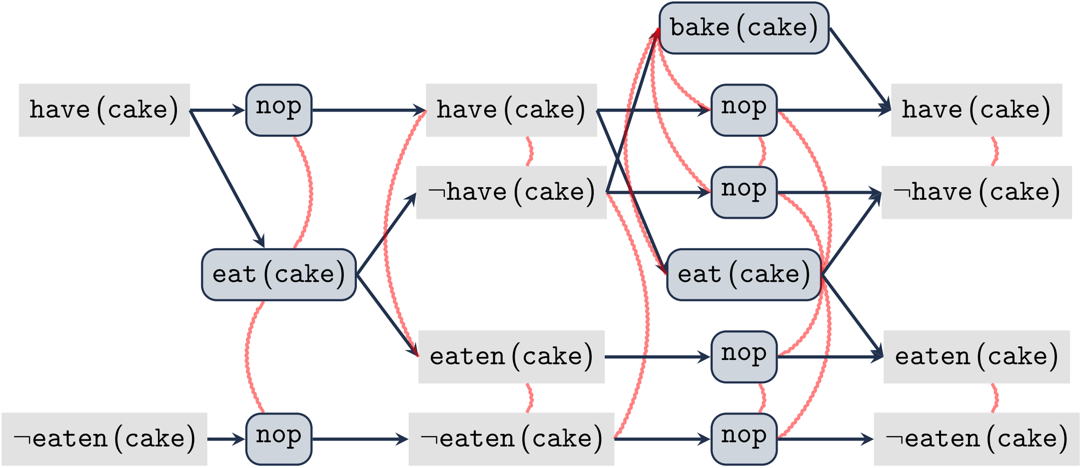
Operators

```
(define (domain cake-domain)
  (:predicates (have ?x)
               (eaten ?x))
  (:action eat  :parameters (?x)
          :precondition (have ?x)
          :effect (and (not (have ?x))
                       (eaten ?x)))
  (:action bake  :parameters (?x)
          :precondition (not (have ?x))
          :effect (and (have ?x))))
```

Facts

```
(define (problem have-and-eat-cake)
  (:domain cake-domain)
  (:objects cake)
  (:init (have cake))
  (:goal (and (have cake)
              (eaten cake))))
```

Example: Cake Planning Graph



Plan Graph Construction

1. Begin with literals for start state
2. Repeatedly add levels:
    2.1 Add persistence (nop) actions for each literal
    2.2 Add feasible actions
    2.3 Mark action mutexes
    2.4 Mark literal mutexes

   until next level is same as prior level (fixpoint)

Action Mutexes

Conflicting Effect: One action's effect negates the other's effect,

- $\texttt{eff}\,(\texttt{eat-cake}) = \texttt{eaten-cake} \wedge \neg\texttt{have-cake}$
- $\texttt{eff}\,(\,\texttt{nop}\,(\,\texttt{have-cake}\,)\,) = \texttt{have-cake}$
- $\neg\,(\texttt{eff}\,(\texttt{eat-cake}) \wedge \texttt{eff}\,(\,\texttt{nop}\,(\,\texttt{have-cake}\,)\,))$

Conflicting Precondition: One action's precondition is mutexed with the other's precondition,

- $\texttt{pre}\,(\texttt{eat-cake}) = \texttt{have-cake}$
- $\texttt{pre}\,(\texttt{bake-cake}) = \neg\texttt{have-cake}$
- $\neg\,(\texttt{pre}\,(\texttt{eat-cake}) \wedge \texttt{pre}\,(\texttt{bake-cake}))$

Interference: One action's effect negates the other's precondition,

- $\texttt{eff}\,(\texttt{eat-cake}) = \neg\texttt{have-cake}$
- $\texttt{pre}\,(\,\texttt{nop}\,(\,\texttt{have-cake}\,)\,) = \texttt{have-cake}$
- $\neg\,(\texttt{eff}\,(\texttt{eat-cake}) \wedge \texttt{nop}\,(\,\texttt{have-cake}\,))$

Literal Mutexes

Negation: One literal is the negation of the other,

- $\neg\left(\texttt{have-cake} \wedge \neg\texttt{have-cake}\right)$
- $\neg\left(\texttt{eaten-cake} \wedge \neg\texttt{eaten-cake}\right)$

Inconsistent Support: Each possible pair of actions to achieve both literals is mutually exclusive

- Step 1:
  - $\texttt{have-cake}^{[1]} \implies \texttt{nop}\left(\texttt{have-cake}\right)^{[0]}$
  - $\texttt{eaten-cake}^{[1]} \implies \texttt{eat-cake}^{[0]}$
  - conflicting effects: $\neg\left(\texttt{nop}\left(\texttt{have-cake}\right)^{[0]} \wedge \texttt{eat-cake}^{[0]}\right)$
- Step 2:
  - $\texttt{have-cake}^{[2]} \implies \left(\texttt{nop}\left(\texttt{have-cake}\right)^{[1]} \vee \texttt{bake-cake}^{[1]}\right)$
  - $\texttt{eaten-cake}^{[2]} \implies \left(\texttt{nop}\left(\texttt{eaten-cake}\right)^{[1]} \vee \texttt{eat-cake}^{[1]}\right)$
  - non-conflicting: $\texttt{bake-cake}^{[1]} \wedge \texttt{nop}\left(\texttt{eaten-cake}\right)^{[1]}$

Exercise: Alternate Cake Domain

Exercise: Air Cargo

Operators

```
(define (domain air-cargo)
  (:predicates (plane ?x) (cargo ?x)
               (airport ?x) (at ?x ?y))
  (:action fly  :parameters (?p ?x ?y)
           :precondition
           (and (plane ?p) (airport ?x) (airport ?y)
                (at ?p ?x))
           :effect (and (not (at ?p ?x)) (at ?p ?y)))
  (:action load  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?a) (at ?p ?a))
           :effect (and (not (at ?c ?a)) (at ?c ?p)))
  (:action unload  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?p) (at ?p ?a))
           :effect (and (not (at ?c ?p)) (at ?c ?a))))
```

Facts

```
(define (problem air)
  (:domain air-cargo)
  (:objects cargo-0 cargo-1
            plane-0 plane-1
            ATL SFO)
  (:init (cargo cargo-0)
         (cargo cargo-1)
         (plane plane-0)
         (plane plane-1)
         (airport ATL)
         (airport SFO)
         (at plane-0 ATL)
         (at plane-1 SFO)
         (at cargo-0 ATL)
         (at cargo-1 SFO))
  (:goal (and (at cargo-0 SFO)
              (at cargo-1 ATL))))
```

Exercise: Air Cargo

Termination of Planning Graph Construction

Theorem

Planning Graphs converge to a fixpoint in a finite number of steps.

Proof Outline

Graph elements increase or decrease monotonically over successive levels:

- **Literals** increase monotonically: Can always persist a literal
- **Actions** increase monotonically: preconditions remain satisfied at successive levels
- **Mutexes** decrease monotonically: mutex at level i holds at all levels below i

Eventually, can add no more literals or actions and can remove no more mutexes. □

Size of Planning Graphs

Theorem

Planning Graphs are polynomial in size of the planning domain.

Proof Outline

- $p = |P|$ propositions, $\ell = 2p$ literals
- $a = |A|$ actions
- Each level:
  - $a + \ell$ nodes
  - max $a * 2\ell$ transition edges (each action to every literal)
  - max $a^2 + \ell^2$ mutex edges (each action/literal mutex with every other)
- Polynomial number of levels due to monotonically increasing/decreasing elements

□

Interpreting of Planning Graphs

Feasibility

- A literal not in the final level (fixpoint) cannot be achieved of plan graph
- Mutexed literals: cannot both hold
  - What if goal literals are mutex at end?

Heuristics

- Cost to achieve literal: level of the graph
- Cost to achieve conjunction:

  Max-level: Maximum cost of arguments
  Level-sum: Sum costs of arguments
  Set-level: Level where all hold

Outline

Planning Graphs
- Construction
- Analysis

Planning with Planning Graphs
- GraphPlan
- GraphPlan+SATPlan (BlackBox)

Overview

1. Successively add levels to the planning graph
2. At each level,
    2.1 If the goals are not mutex, attempt to extract a plan from the graph
    2.2 If no plan can be extracted, continue growing the graph

Plan Extraction: GraphPlan Proper

Backwards Search from Final Level:

1. Start from last level from graph
2. Select conflict free actions from predecessor level to achieve current goal
3. New goal is precondition of selected actions
4. Repeat

Plan Extraction: GraphPlan + SATPlan (BlackBox Planner)

1. Construct planning graph
2. Convert planning graph to Boolean formula
3. Check SAT
4. If UNSAT, repeat

BlackBox vs. SATPlan proper: mutex information

SATPlan Encoding

Transition Function

Operator Encoding: Selected operator's preconditions and effects must hold:

$$o_i^{[k]} \implies \left(\overbrace{\mathrm{pre}(o_i)^{[k]}}^{\text{precondition at step } k} \land \overbrace{\mathrm{eff}(o_i)^{[k+1]}}^{\text{effect at step } k+1} \right)$$

Operator Exclusion: One operator per step:

$$o_i^{[k]} \implies \left(\neg o_0^{[k]} \land \neg o_{(i-1)}^{[k]} \land \neg o_{(i+1)}^{[k]} \land \neg o_m^{[k]}\right)$$

Frame Axioms: Each proposition p is unchanged unless set by an effect:

$$\left(p^{[k]} = p^{[k+1]}\right) \lor \overbrace{\left(o_j^{[k]} \lor \ldots \lor o_\ell^{[k]}\right)}^{\text{operators changing } p}$$

SATPlan vs. Blackbox Encoding

Transition Function

SATPlan Operator Exclusion: One operator per step:

$$o_i^{[k]} \implies \left(\neg o_0^{[k]} \wedge \neg o_{(i-1)}^{[k]} \wedge \neg o_{(i+1)}^{[k]} \wedge \neg o_m^{[k]}\right)$$

BlackBox Operator Exclusion: Multiple (non-conflicting) operators per step:

$$\left\{ o_i^{[k]} \implies \neg o_j^{[k]} \;\middle|\; o_i^{[k]} \text{ and } o_j^{[k]} \text{ are mutex} \right\}$$

Blackbox Encoding

State Mutexes

$$\left\{ \neg \left({\ell_i}^{[k]} \wedge {\ell_j}^{[k]} \right) \;\middle|\; {\ell_i}^{[k]} \text{ and } {\ell_j}^{[k]} \text{ are mutex} \right\}$$

Additional constraint restricts search space.

Summary

Planning Graphs

- Construction
- Analysis

Planning with Planning Graphs

- GraphPlan
- GraphPlan+SATPlan (BlackBox)

References

Textbook: Russell & Norvig.

- Ch 10.3 Planning Graphs