

# Artificial Intelligence Programming

## Planning

Chris Brooks

Department of Computer Science  
University of San Francisco

## Planning

- So far, we've looked at how to infer new knowledge about a problem, but we haven't talked about using this to select actions to take.
- *Planning* is the task of selecting a sequence of actions that will achieve a goal.
- Challenges:
  - Representing actions
  - Dealing with large search spaces
  - Taking advantage of problem decomposition

## The Planning Problem

- Planning combines search and knowledge representation.
- Uses a logical formalism to describe states and actions
- Uses heuristic search to select actions to take
- Takes advantage of the fact that planning problems are often decomposable.

Department of Computer Science — University of San Francisco — p.1/77

Department of Computer Science — University of San Francisco — p.1/77

## The Planning Problem

- For example, a "Travel to Hawaii" plan can be broken into:
  - Buying tickets stage
  - Packing stage
  - Getting to the airport stage
  - Flying on the plane stage
- These problems are independent - the order in which I pack things shouldn't affect how I get to the airport.
- This means that we don't need to solve these problems in the order they'll be executed.

Department of Computer Science — University of San Francisco — p.3/77

## Applications of Planning

- Planning has been one of the most successful AI technologies.
  - NASA's Remote Agent, MARS rovers
  - Autopilots
  - Home Health Assistants
  - Logistics coordination
  - Beer factory production

Department of Computer Science — University of San Francisco — p.4/77

## Representation

- Planning operators look very much like rules and facts.
- *States* are conjunctions of positive literals and relations between literals.
  - How does this compare to our previous representation of states in search?
- No variables or functions allowed in states.
  - $At(Brooks, Airport)$
  - $In(HawaiianShirt, Suitcase) \wedge In(Toothbrush, Suitcase) \wedge Has(Brooks, Ticket)$

Department of Computer Science — University of San Francisco — p.4/77

## Representation

- A goal is a partially specified state.
  - A conjunction of positive literals, with not every state feature specified.
  - e.g.  $In(Brooks, Hawaii) \wedge Has(Brooks, Suitcase) \wedge In(HawaiianShirt, Suitcase)$
  - (no mention of where my ticket is in the goal state)

## Representing Actions

- An action is specified in terms of *preconditions* and *effects*.
- Preconditions indicate what must hold in order for the action to be taken.
- Effects indicate how the action changes the world.
- Both may contain variables.
  - Action :  $BuyTicket(person, price, ticket)$   
Preconditions :  
 $Costs(ticket, price) \wedge HasMoney(person, price)$   
Effects :  $Has(ticket, person) \wedge \neg Has(person, price)$
  - Action :  $DriveToAirport(person, loc)$   
Preconditions :  
 $At(person, loc) \wedge Different(loc, Airport)$   
Effects :  $\neg At(person, loc) \wedge At(person, Airport)$

## STRIPS assumptions

- Preconditions are positive conjunctions
- Every literal not mentioned remains unchanged.
- Goals can only contain ground literals
  - Can't have a goal like  
 $\exists loc At(Brooks, loc) \wedge Warm(loc)$
- Goals are conjunctions
- These assumptions limit the sorts of problems that can be solved, but make planning algorithms simpler and more efficient.

## Formulating a Planning Problem

- Specify actions
  - $RemoveTire(tire)$   
Preconditions :  $On(tire, Axle)$   
Effect :  
 $Clear(Axle) \wedge \neg On(tire, Axle) \wedge On(tire, Ground)$
  - $PutOnTire(tire)$   
Preconditions :  $On(tire, Ground) \wedge Clear(Axle)$   
Effect :  
 $On(tire, Axle) \wedge \neg On(tire, Ground) \wedge \neg Clear(Axle)$
  - $TakeFromTrunk(tire)$   
Preconditions :  $In(tire, Trunk)$   
Effect :  $On(tire, Ground) \wedge \neg In(tire, Trunk)$
  - $PutInTrunk(tire)$   
Preconditions :  $On(tire, Ground)$   
Effect :  $In(tire, Trunk) \wedge \neg On(tire, Ground)$

## Solving a Planning Problem

- Now that we have a formalization for the problem, we can try to apply our standard search techniques.
- We can search forward from the initial state to the goal state.
- At each state, consider what actions are possible.
- Each potential action generates a new state.
- This is called progression planning.
  - Very similar to forward chaining.
- Problem: Lots of irrelevant actions are considered.
- Doesn't scale to complex domains.

## Solving a Planning Problem

- We can also work backward from the goal to the initial state.
- This is called regression planning.
- Look for actions that achieve one or more goal criteria.
- Algorithm is similar to backward chaining.
- Still doesn't scale effectively.
- Some problems cannot be solved using pure regression planning.
- Doesn't allow you to take advantage of problem decomposition.

## Partial-order Planning

- One problem with progression and regression planning is that they search for linear sequences of actions.
  - Often, subgoals can be solved in more than one order.
  - It really doesn't matter whether I buy my ticket before I pack.
- Partial-order planning solves subgoals independently (as much as possible) and then combines subplans.
- No need to select steps in chronological order.
- Example: in planning my trip, I know I'll need to buy a ticket, pack, and get to the airport. I'll figure out how to do each of those, then decide on an order later.
- This is called a *least commitment* strategy.

## Partial-order Planning

- Partial-order planning searches in the space of *partially-completed plans*.
- This is different from A\*, BFS, regression planning, etc, which search in a space of *states*
  - In search, our successor function returned all possible actions and their resulting states.
  - In planning, our successor function returns all more highly refined plans.
  - In other words, we only consider actions that can be seen to help us reach our goal.

## Partial-order Planning

- A plan has four components:
  - A set of actions that make up the steps of the plan.
  - A set of ordering constraints that indicate a sequence on actions.
    - For example,  $PackSuitcase \prec GoToAirport$
  - A set of causal links that indicate one action achieves the precondition of another.
    - For example,  $BuyTicket \rightarrow HasTicket \ TakeFlight$
  - A set of open preconditions. These are preconditions not achieved by any action in the current plan.

## Partial-order Planning

- A *consistent plan* is one in which there are no cycles in the ordering constraints.
- A consistent plan with no open preconditions is a *solution*.
- We can then *linearize* this plan to get a sequence of actions.
- Any linearization of a partial-order plan will reach the goal state.
  - Some linearizations might be more efficient than others.

## POP algorithm

- Initial plan contains  $Start, Finish, Start \prec Finish$
- While (no solution)
  - Select an open precondition  $p$  on an action  $B$
  - Find each action  $A$  that satisfies that precondition; generate a new plan for each action.
  - for each of these plans:
    - Add ordering constraints  $A \prec B$ , plus causal links  $A \rightarrow^p B$
    - Resolve any conflicts between causal links. If no consistent plan exists, discard.

## POP example

- Problem Description
  - $Initial : On(FlatTire, Axle) \wedge In(SpareTire, Trunk)$
  - $Goal : On(SpareTire, Axle) \wedge In(FlatTire, Trunk)$
  - $RemoveTire(tire)$ 
    - Preconditions :  $On(tire, Axle)$
    - Effect :  $Clear(Axle) \wedge On(tire, Ground)$
  - $PutOnTire(tire)$ 
    - Preconditions :  $On(tire, Ground) \wedge Clear(Axle)$
    - Effect :  $On(tire, Axle) \wedge \neg On(tire, Ground) \wedge \neg Clear(Axle)$
  - $TakeFromTrunk(tire)$ 
    - Preconditions :  $In(tire, Trunk)$
    - Effect :  $On(tire, Ground) \wedge \neg In(tire, Trunk)$
  - $PutInTrunk(tire)$ 
    - Preconditions :  $On(tire, Ground)$
    - Effect :  $In(tire, Trunk) \wedge \neg On(tire, Ground)$

## POP example

- Open preconditions:  
 $On(SpareTire, Axle) \wedge In(FlatTire, Trunk)$
- Pick  $On(SpareTire, Axle)$ , select  $PutOnTire$ .
  - Add causal links, ordering constraints between  $PutOnTire$  and  $Finish$ .
- New preconditions:  
 $In(FlatTire, Trunk), On(SpareTire, Ground), Clear(Axle)$
- Pick  $In(FlatTire, Trunk)$ , select  $PutInTrunk$ .
  - Add causal links, ordering constraints between  $PutInTrunk$  and  $Finish$ .
- New Preconditions:  
 $On(SpareTire, Ground), Clear(Axle), On(FlatTire, Ground)$

Department of Computer Science — University of San Francisco — p.18/77

## POP example

- Pick  $On(SpareTire, Ground)$ , select  $TakeFromTrunk$ 
  - Add ordering constraints and causal links between  $TakeFromTrunk$  and  $PutOnTire$ , also between  $Start$  and  $TakeFromTrunk$ .
- New Preconditions:  $Clear(Axle), On(FlatTire, Ground)$
- Select  $RemoveTire$ 
  - Add ordering constraints and causal links between  $RemoveTire$  and  $PutInTrunk$
- We have a solution.

Department of Computer Science — University of San Francisco — p.19/77

## POP heuristics

- The trickiest part of POP involves choosing a precondition to try to satisfy.
- It can be hard to estimate how “close” a plan is to a solution.
- A common heuristic:
  - Most-constrained variable - select the open precondition that can be satisfied in the fewest number of ways.
  - Intuition: Locate “bottlenecks” early on.

Department of Computer Science — University of San Francisco — p.20/77

## More sophisticated planning

- STRIPS was developed in the late 60s.
- Since then, *many* advances have been made in planning.
- Expressivity:
  - Disjunctive effects
  - Existential goals
  - Time (STRIPS assumes actions are instantaneous)

Department of Computer Science — University of San Francisco — p.21/77

## More sophisticated planning

- STRIPS was developed in the late 60s.
- Since then, *many* advances have been made in planning.
- Scalability
  - A big challenge in planning was scaling to real-world domains
  - Hierarchical planning
  - Precompiled plan libraries

Department of Computer Science — University of San Francisco — p.22/77

## More sophisticated planning

- STRIPS was developed in the late 60s.
- Since then, *many* advances have been made in planning.
- Integration with execution
  - STRIPS assumes the plan is created offline and will execute without errors.
  - Online planning, replanning, plan repair is essential in complex environments.
  - Integration with real-time execution environments.
  - Multi-agent planning

Department of Computer Science — University of San Francisco — p.23/77

## Weaknesses of planning

- STRIPS-style planning works with *qualitative information*.
  - Information easily represented as logical predicates.
  - Difficult to incorporate probabilities.
- Representational assumptions
  - Trade off efficiency for representational power.
  - Offline planning, plan compilation, plan libraries can help with this.
- After the midterm, we'll look at how to integrate quantitative uncertainty into planning (Markov decision processes).