Planning

Based on slides prepared by Tom Lenaerts
SWITCH, Vlaams Interuniversitair Instituut voor Biotechnologie

Modifications by Jacek.Malec@cs.lth.se
Original slides can be found at http://aima.cs.berkeley.edu

LUND UNIVERSITY

Vrije Universiteit Brussel

1

Planning

The Planning problem
Planning with State-space search
Planning with propositional logic
Partial-order planning
Planning graphs
Analysis of planning approaches

2

What is Planning

Generate sequences of actions to perform tasks and achieve objectives.
- States, actions and goals

Search for solution over abstract space of plans.

Classical planning environment: fully observable, deterministic, finite, static and discrete.

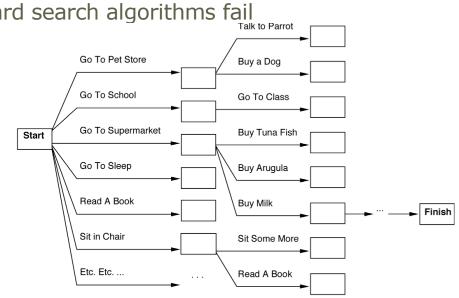
Assists humans in practical applications
- design and manufacturing
- military operations
- games
- space exploration
- transport and logistics

3

Why not standard search?

Consider the task get milk, bananas and a cordless drill

Standard search algorithms fail



4

Difficulty of real world problems

Assume a problem-solving agent using some search method …

- Which actions are relevant?
  - Exhaustive search vs. backward search
- What is a good heuristic functions?
  - Good estimate of the cost of the state?
  - Problem-dependent vs, -independent
- How to decompose the problem?
  - Most real-world problems are ***nearly*** decomposable.

Planning language

What is a good language?

- Expressive enough to describe a wide variety of problems.
- Restrictive enough to allow efficient algorithms to operate on it.
- Planning algorithm should be able to take advantage of the logical structure of the problem.

STRIPS and PDDL

General language features

Representation of states

- Decompose the world in logical conditions and represent a state as a *conjunction of positive literals.*
  - Propositional literals: *Safe* ∧ *HasGold*
  - FO-literals (grounded and function-free): *At(Plane1, Copenhagen)* ∧ *At(Plane2, Stockholm)*
- Closed world assumption

Representation of goals

- Partially specified state and represented as a *conjunction of positive ground literals*
- A goal is *satisfied* if the state contains all literals in goal.

General language features

Representations of actions

- Action = PRECOND + EFFECT

Action(Fly(p, from, to),
PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
EFFECT: ¬AT(p, from) ∧ At(p, to))

= action schema (p, from, to need to be instantiated)
  - Action name and parameter list
  - Precondition (conj. of function-free literals)
  - Effect (conjunction of function-free literals and P is True and not P is false)
- Add-list vs delete-list in Effect

Language semantics?

How do actions affect states?

- An action is applicable in any state that satisfies the precondition.
- For first order action schema applicability involves a substitution θ for the variables in the PRECOND.

  At(P1, ARN) ∧ At(P2, CPH) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(ARN) ∧ Airport(CPH)

  Satisfies : *At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)*

  With θ ={*p/P1, from/ARN, to/CPH*}

  Thus the action is applicable.

Language semantics?

The result of executing action a in state s is the state s'

- s' is same as s except
  - Any positive literal *P* in the effect of *a* is added to *s'*
  - Any negative literal ¬*P* is removed from *s'*

  EFFECT: ¬AT(p, from) ∧ At(p, to):

  At(P1, CPH) ∧ At(P2, CPH) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(ARN) ∧ Airport(CPH)

- STRIPS assumption: (avoids representational frame problem)

  every literal NOT in the effect remains unchanged

Expressiveness and extensions

STRIPS is simplified

- Important limit: function-free literals
  - Allows for propositional representation
  - Function symbols lead to infinitely many states and actions

Expressiveness extension: Planning Domain Description Language (PDDL)

Action(Fly(p: Plane, from: Airport, to: Airport),
PRECOND: At(p, from) ∧ (from ≠ to)
EFFECT: ¬At(p, from) ∧ At(p, to))

Standardization : *now (since 2008) in its 3.1 version*

Example: air cargo transport

Init(At(C1, CPH) ∧ At(C2, ARN) ∧ At(P1, CPH) ∧ At(P2, ARN) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(ARN) ∧ Airport(CPH))

Goal(At(C1, ARN) ∧ At(C2, CPH))

Action(Load(c, p, a)
PRECOND: *At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)*
EFFECT: *¬At(c, a) ∧ In(c, p))*
Action(Unload(c, p, a)
PRECOND: *In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)*
EFFECT: *At(c, a) ∧ ¬In(c, p))*
Action(Fly(p, from, to)
PRECOND: *At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)*
EFFECT: *¬At(p, from) ∧ At(p, to))*

[Load(C1, P1 ,CPH), Fly(P1, CPH, ARN), Unload(C1, P1, ARN), Load(C2, P2, ARN), Fly(P2, ARN, CPH), Unload(C2, P2, CPH)]

Example: Spare tire problem

Init(At(Flat, Axle) ∧ At(Spare, trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk)
  PRECOND: *At(Spare, Trunk)*
  EFFECT: *¬At(Spare, Trunk) ∧ At(Spare, Ground))*
Action(Remove(Flat, Axle)
  PRECOND: *At(Flat, Axle)*
  EFFECT: *¬At(Flat, Axle) ∧ At(Flat, Ground))*
Action(PutOn(Spare, Axle)
  PRECOND: *At(Spare, Groundp) ∧ ¬At(Flat, Axle)*
  EFFECT: *At(Spare, Axle) ∧ ¬At(Spare ,Ground))*
Action(LeaveOvernight
  PRECOND:
  EFFECT: *¬At(Spare, Ground) ∧ ¬At(Spare, Axle) ∧ ¬At(Spare, trunk) ∧ ¬At(Flat, Ground) ∧ ¬At(Flat, Axle))*

This example goes beyond STRIPS: negative literal in pre-condition (PDDL description)

13

Example: Blocks world

Init(On(A, Table) ∧ On(B, Table) ∧ On(C, Table) ∧ Block(A) ∧ Block(B) ∧ Block(C) ∧ Clear(A) ∧ Clear(B) ∧ Clear(C))

Goal(On(A, B) ∧ On(B, C))

Action(Move(b, x, y)
  PRECOND: *On(b, x) ∧ Clear(b) ∧ Clear(y) ∧ Block(b) ∧ (b ≠ x) ∧ (b ≠ y) ∧ (x ≠ y)*
  EFFECT: *On(b, y) ∧ Clear(x) ∧ ¬On(b, x) ∧ ¬Clear(y))*
Action(MoveToTable(b, x)
  PRECOND: *On(b, x) ∧ Clear(b) ∧ Block(b) ∧ (b ≠ x)*
  EFFECT: *On(b, Table) ∧ Clear(x) ∧ ¬On(b, x))*

14

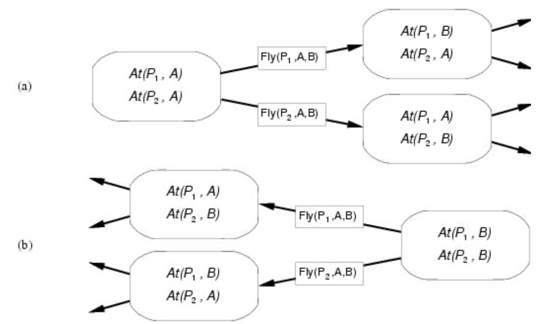
Planning with state-space search

Both forward and backward search possible
Progression planners
- forward state-space search
- Consider the effect of all possible actions in a given state

Regression planners
- backward state-space search
- To achieve a goal, what must have been true in the previous state.

15

Progression and regression



16

Progression algorithm

Formulation as state-space search problem:

- Initial state = initial state of the planning problem
  - Literals not appearing are false
- Actions = those whose preconditions are satisfied
  - Add positive effects, delete negative
- Goal test = does the state satisfy the goal
- Step cost = each action costs 1

No functions … any graph search that is complete is a complete planning algorithm.

- E.g. A*

Inefficient:

- (1) irrelevant action problem
- (2) good heuristic required for efficient search

17

Regression algorithm

How to determine predecessors?

- What are the states from which applying a given action leads to the goal?

  Goal state = *At(C1, B) ∧ At(C2, B) ∧ … ∧ At(C20, B)*
  Relevant action for first conjunct: *Unload(C1, p, B)*
  Works only if pre-conditions are satisfied.
  Previous state= *In(C1, p) ∧ At(p, B) ∧ At(C2, B) ∧ … ∧ At(C20, B)*
  Subgoal At(C1, B) should not be present in this state.

Actions must not undo desired literals (consistent)

Main advantage: only relevant actions are considered.

- Often much lower branching factor than forward search.

18

Regression algorithm

General process for predecessor construction

- Give a goal description G
- Let A be an action that is relevant and consistent
- The predecessors are as follows:
  - Any positive effects of A that appear in G are deleted.
  - Each precondition literal of A is added , unless it already appears.

Any standard search algorithm can be added to perform the search.

Termination when predecessor is satisfied by initial state.

- In FO case, satisfaction might require a substitution.

19

Heuristics for state-space search

Neither progression or regression are very efficient without a good heuristic.

- How many actions are needed to achieve the goal?
- Exact solution is NP hard, find a good estimate

Two approaches to find admissible heuristic:

- The optimal solution to the relaxed problem.
  - Remove all preconditions from actions
- The subgoal independence assumption:

  The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

20

Planning with propositional logic

Planning can be done by proving theorem in situation calculus.
Here: test the *satisfiability* of a logical sentence:

initial state ∧ *all possible action descriptions* ∧ *goal*

Sentence contains propositions for every action occurrence.
- A model will assign true to the actions that are part of the correct plan and false to the others
- An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that the goal is true.
- If the planning is unsolvable the sentence will be unsatisfiable.

Partial-order planning

Progression and regression planning are *totally ordered plan search* forms.
- They cannot take advantage of problem decomposition.
  - Decisions must be made on how to sequence actions on all the subproblems

Least commitment strategy:
- Delay choice during search

Shoe example

Goal(RightShoeOn ∧ LeftShoeOn)

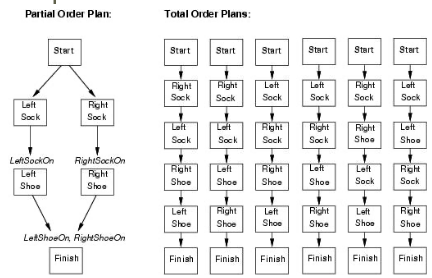
Init()

Action(RightShoe, PRECOND: RightSockOn EFFECT: RightShoeOn)
Action(RightSock, PRECOND: EFFECT: RightSockOn)
Action(LeftShoe, PRECOND: LeftSockOn EFFECT: LeftShoeOn)
Action(LeftSock, PRECOND: EFFECT: LeftSockOn)

Planner: combine two action sequences
(1) leftsock, leftshoe
(2) rightsock, rightshoe

Partial-order planning(POP)

Any planning algorithm that can place two actions into a plan without stating which comes first is a PO plan.



POP as a search problem

States are (mostly unfinished) plans.

- The empty plan contains only start and finish actions.

Each plan has 4 components:

- A set of actions (steps of the plan)
- A set of ordering constraints: A < B (A before B)
  - Cycles represent contradictions.
- A set of causal links $A \xrightarrow{p} B$
  - The plan may not be extended by adding a new action C that conflicts with the causal link. (if the effect of C is ¬p and if C could come after A and before B)
- A set of open preconditions.
  - If precondition is not achieved by action in the plan.

Example of final plan

Actions={Rightsock, Rightshoe, Leftsock, Leftshoe, Start, Finish}

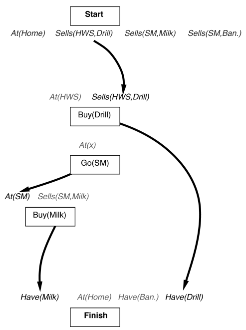
Orderings={Rightsock < Rightshoe; Leftsock < Leftshoe}

Links={Rightsock->Rightsockon -> Rightshoe, Leftsock->Leftsockon-> Leftshoe, Rightshoe->Rightshoeon->Finish, …}

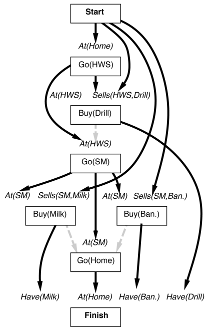
Open preconditions={}

Shopping list example

Shopping list example



Shopping list example



29

POP as a search problem

A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.

A consistent plan with no open preconditions is a *solution*.

A partial order plan is executed by repeatedly choosing *any* of the possible next actions.

- This flexibility is a benefit in non-cooperative environments;
- Gives rise to emergent behaviours.

30

Solving POP

Assume propositional planning problems:

- The initial plan contains *Start* and *Finish*, the ordering constraint *Start* < *Finish*, no causal links, all the preconditions in *Finish* are open.
- Successor function :
  - picks one open precondition *p* on an action *B* and
  - generates a successor plan for every possible consistent way of choosing action *A* that achieves *p*.
- Test goal

31

Enforcing consistency

When generating successor plan:

- The causal link *A->p->B* and the ordering constraint A < B is added to the plan.
  - If A is new also add start < A and A < B to the plan
- Resolve conflicts between new causal link and all existing actions
- Resolve conflicts between action A (if new) and all existing causal links.

32

Process summary

Operators on partial plans

- Add link from existing plan to open precondition.
- Add a step to fulfill an open condition.
- Order one step w.r.t another to remove possible conflicts

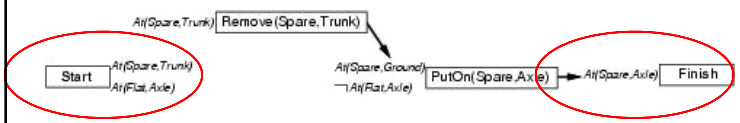
Gradually move from incomplete/vague plans to complete/correct plans

Backtrack if an open condition is unachievable or if a conflict is irresolvable.

Example: Spare tire problem

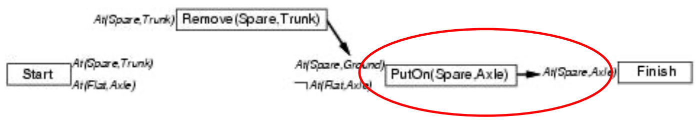
Init(At(Flat, Axle) ∧ At(Spare, trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk)
  PRECOND: *At(Spare, Trunk)*
  EFFECT: *¬At(Spare, Trunk) ∧ At(Spare, Ground))*
Action(Remove(Flat, Axle)
  PRECOND: *At(Flat, Axle)*
  EFFECT: *¬At(Flat, Axle) ∧ At(Flat, Ground))*
Action(PutOn(Spare, Axle)
  PRECOND: *At(Spare, Groundp) ∧ ¬At(Flat, Axle)*
  EFFECT: *At(Spare ,Axle) ∧ ¬At(Spare, Ground))*
Action(LeaveOvernight
  PRECOND:
  EFFECT: *¬At(Spare, Ground) ∧ ¬At(Spare, Axle) ∧ ¬At(Spare, trunk) ∧ ¬At(Flat, Ground) ∧ ¬At(Flat, Axle))*

Solving the problem



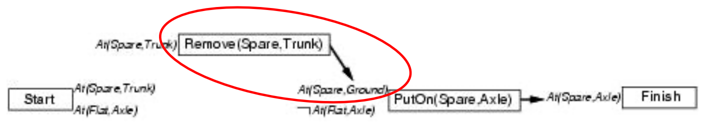
Initial plan: Start with EFFECTS and Finish with PRECOND.

Solving the problem



Initial plan: Start with EFFECTS and Finish with PRECOND.
Pick an open precondition: *At(Spare, Axle)*
Only *PutOn(Spare, Axle)* is applicable
Add causal link: $PutOn(Spare, Axle) \xrightarrow{At(Spare,Axle)} Finish$
Add constraint : *PutOn(Spare, Axle) < Finish*

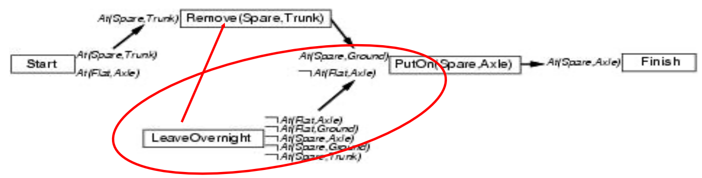
Solving the problem



Pick an open precondition: *At(Spare, Ground)*
Only *Remove(Spare, Trunk)* is applicable
Add causal link: $Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$
Add constraint : *Remove(Spare, Trunk) < PutOn(Spare,Axle)*

37

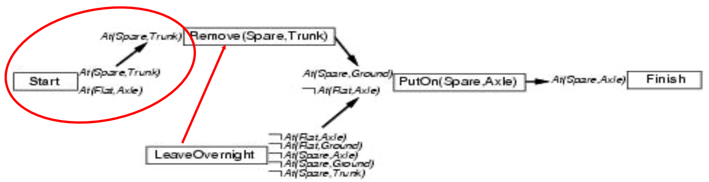
Solving the problem



Pick an open precondition: ¬*At(Flat, Axle)*
LeaveOverNight is applicable
conflict: *LeaveOverNight* also has the effect ¬ *At(Spare,Ground)*
$Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$
To resolve, add constraint : *LeaveOverNight < Remove(Spare, Trunk)*

38

Solving the problem



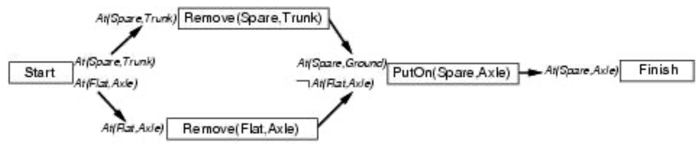
Pick an open precondition: *At(Spare, Trunk)*
Only *Start* is applicable
Add causal link: $Start \xrightarrow{At(Spare,Trunk)} Remove(Spare,Trunk)$
Conflict: of causal link with effect ¬*At(Spare,Trunk)* in *LeaveOverNight*

- *No re-ordering solution possible.*

backtrack

39

Solving the problem



Remove *LeaveOverNight*, *Remove(Spare, Trunk)* and causal links
Repeat step with Remove(Spare,Trunk)
Add also RemoveFlatAxle and finish

40

Some details …

What happens when a first-order representation that includes variables is used?

- Complicates the process of detecting and resolving conflicts.
- Can be resolved by introducing inequality constraint.

CSP's most-constrained-variable heuristic can be used for planning algorithms to select a PRECOND.

Analysis of planning approach

Planning is an area of great interest within AI

- Search for solution
- Constructively prove existence of a solution

Biggest problem is the combinatorial explosion in states.

Efficient methods are under research

- E.g. divide-and-conquer

Planning vs. scheduling

Classical planning:

What to do? In what order?

But not:

How long? When? Using what resources?

Normally:

Plan first, schedule later.

Representation

Job-shop scheduling problem:

- A set of jobs
- Each job is a collection of ACTIONS with some ORDERING CONSTRAINTS
- Each action has a DURATION and a set of RESOURCE CONSTRAINTS

resources may be CONSUMABLE or REUSABLE

Solution:

Start times for all actions, obeying all constraints