Foundations of Artificial Intelligence

10. Action Planning

Solving Logically Specified Problems using a General Problem Solver

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

June 7, 2019

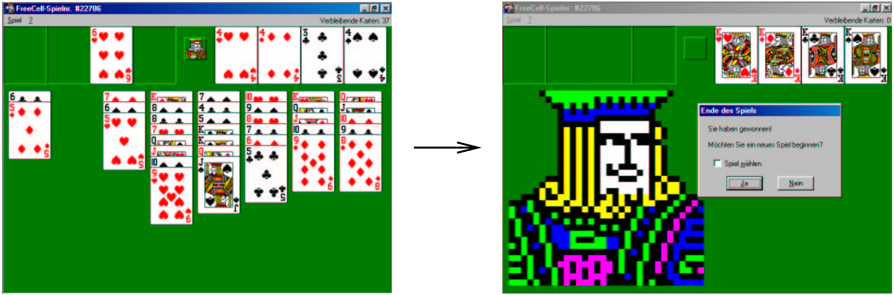
Contents

Lecture Overview

Planning

- Planning is the art and practive of thinking before acting [Haslum]
- Planning is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior.
- The outcome is usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

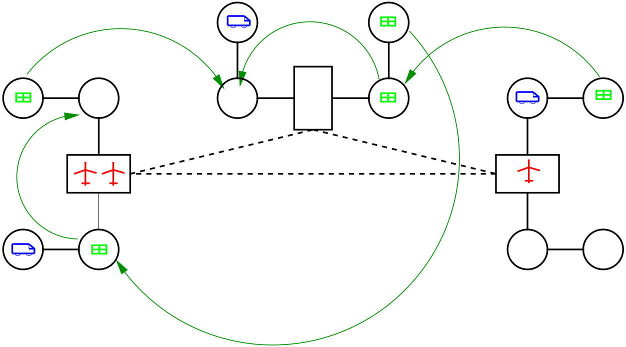
Given a current state, a set of possible actions, a specification of the goal conditions, which plan transforms the current state into a goal state?



Search through possible configuration

Another Planning Task: *Logistics*

Given a road map, and a number of trucks and airplanes, make a plan to transport objects from their start to their goal destinations.



Action Planning is not . . .

- Problem solving by search, where we describe a problem by a state space and then implement a program to search through this space
  - in action planning, we specify the problem declaratively (using logic) and then solve it by a general planning algorithm DON'T KNOW TO IMPLEMENT.
- Program synthesis, where we generate programs from specifications or examples
  - in action planning we want to solve just one instance and we have only very simple action composition (i.e., sequencing, perhaps conditional and iteration)
- Scheduling, where all jobs are known in advance and we only have to fix time intervals and machines
  - instead we have to find the right actions and to sequence them
- Of course, there is interaction with these areas!

Lecture Overview

Domain-Independent Action Planning

- Start with a declarative specification of the planning problem
- Use a domain-independent planning system to solve the planning problem
- Domain-independent planners are generic problem solvers
- Issues:
  - Good for evolving systems and those where performance is not critical
  - Running time should be comparable to specialized solvers
  - Solution quality should be acceptable
  - . . . at least for all the problems we care about

Planning as Logical Inference

Planning can be elegantly formalized with the help of the *situation calculus*.

Initial state:
$At(truck1, loc1, s_0) \wedge At(package1, loc3, s_0)$

Operators (successor-state axioms):
$\forall a, s, l, p, t\ At(t, p, Do(a, s)) \Leftrightarrow \{a = Drive(t, l, p) \wedge Poss(Drive(t, l, p), s)$
$\vee At(t, p, s) \wedge (a \neq \neg Drive(t, p, l, s) \vee \neg Poss(Drive(t, p, l), s))\}$

Goal conditions (query):
$\exists s\ At(package1, loc2, s)$

The constructive proof of the existential query (computed by a automatic theorem prover) delivers a plan that does what is desired. Can be quite inefficient! only for small samples.

The Basic STRIPS Formalism

STRIPS: STanford Research Institute Problem Solver

- $\mathcal{S}$ is a *first-order vocabulary* (predicate and function symbols) and $\Sigma_{\mathcal{S}}$ denotes the set of *ground atoms* over the signature (also called **facts** or **fluents**).
- $\Sigma_{\mathcal{S},\mathbf{V}}$ is the set of atoms over $\mathcal{S}$ using variable symbols from the set of variables $\mathbf{V}$.

  ground atoms.
- A **first-order STRIPS state** S is a subset of $\Sigma_{\mathcal{S}}$ denoting a *complete theory* or *model* (using CWA).
- A **planning task** (or **planning instance**) is a 4-tuple $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$, where
  - $\mathbf{O}$ is a set of **operator** (or *action types*)
  - $\mathbf{I} \subseteq \Sigma_{\mathcal{S}}$ is the **initial state**
  - $\mathbf{G} \subseteq \Sigma_{\mathcal{S}}$ is the **goal specification**
- No domain constraints (although present in original formalism)

Operators, Actions & State Change

Effects: set of literals

- **Operator:**

$$o = \langle para, pre, eff \rangle,$$

  with $para \subseteq \mathbf{V}$, $pre \subseteq \Sigma_{\mathcal{S},\mathbf{V}}$, $eff \subseteq \Sigma_{\mathcal{S},\mathbf{V}} \cup \neg\Sigma_{\mathcal{S},\mathbf{V}}$ (element-wise negation) and all variables in pre and eff are listed in $para$.
  Also: $pre(o), eff(o)$.
  eff^{+} = positive effect literals
  eff^{-} = negative effect literals

- **Operator instance** or **action**: Operator with empty parameter list (*instantiated schema!*)

- **State change** induced by action:

$$App(S, o) = \begin{cases} S \cup eff^{+}(o) - \neg eff^{-}(o) & \text{if } pre(o) \subseteq S \text{ \& } eff(o) \text{ is cons.} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example Formalization: *Logistics*

- Logical atoms: $at(O,L)$, $in(O,V)$, $airconn(L1,L2)$, $street(L1,L2)$, $plane(V)$, $truck(V)$
- Load into truck: $load$
  - Parameter list: (O,V,L)
  - Precondition: $at(O,L), at(V,L), truck(V)$
  - Effects: $\neg at(O,L), in(O,V)$

    no longer   now
- Drive operation: $drive$
  - Parameter list: $(V,L1,L2)$
  - Precondition: $at(V,L1), truck(V), street(L1,L2)$
  - Effects: $\neg at(V,L1), at(V,L2)$
- …
- Some constant symbols: $v1, s, t$ with $truck(v1)$ and $street(s,t)$
- Action: $drive(v1,s,t)$

Plans & Successful Executions

- A plan Δ is a sequence of actions
- State resulting from **executing a plan**:

$$
\begin{aligned}
Res(S, \langle\rangle) &= S \\
Res(S, (o; \Delta)) &= \begin{cases} Res(App(S, o), \Delta) & \text{if } App(S, o) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}
\end{aligned}
$$

- **Plan Δ is successful** or solves a planning task if $Res(\mathbf{I}, \Delta)$ is defined and $\mathbf{G} \subseteq Res(\mathbf{I}, \Delta)$.

A Small Logistics Example

Initial state: $S \;=\; \left\{ \begin{array}{l} at(p1,c), at(p2,s), at(t1,c), \\ at(t2,c), street(c,s), street(s,c) \end{array} \right\}$

Goal: $\mathbf{G} \;=\; \{\; at(p1,s), at(p2,c) \;\}$

in

to

Successful plan: $\Delta \;=\; \begin{array}{l} \langle load(p1,t1,c), drive(t1,c,s), \\ unload(p1,t1,s), load(p2,t1,s), \\ drive(t1,s,c), unload(p2,t1,c) \rangle \end{array}$

Other successful plans are, of course, possible

Simplifications: DATALOG- and Propositional STRIPS

- STRIPS as described above allows for unrestricted first-order terms, i.e., arbitrarily nested **function terms**

→ **Infinite state space**

- Simplification: No function terms (only 0-ary = constants)

→ **DATALOG-STRIPS**

- Simplification: No variables in operators (= actions)

→ **Propositional STRIPS**

- Propositional STRIPS used in planning algortihms nowadays (but specification is done using DATALOG-STRIPS)

Beyond STRIPS

Even when keeping all the restrictions of classical planning, one can think of a number of extensions of the planning language.

- General logical formulas as preconditions: Allow all Boolean connectors and quantification
- Conditional effects: Effects that happen only if some additional conditions are true. For example, when pressing the accelerator pedal, the effects depends on which gear has been selected (no, reverse, forward).
- Multi-valued state variables: Instead of 2-valued Boolean variables, multi-valued variables could be used
- Numerical resources: Resources (such as fuel or time) can be effected and be used in preconditions
- Durative actions: Actions can have duration and can be executed concurrently
- Axioms/Constraints: The domain is not only described by operators, but also by additional laws

PDDL: The Planning Domain Description Language

- Since 1998, there exists a bi-annual scientific competition for action planning systems.
- In order to have a common language for this competition, PDDL has been created (originally by Drew McDermott)
- Meanwhile, version 3.1 (IPC-2011) with most of the features mentioned – and many sub-dialects and extensions.
- Sort of "standard" by now.

PDDL Logistics Example

```
(define (domain logistics)
    (:types truck airplane - vehicle
            package vehicle - physobj
            airport location - place
            city place physobj - object)

    (:predicates (in-city ?loc - place ?city - city)
            (at ?obj - physobj ?loc - place)
            (in ?pkg - package ?veh - vehicle))

(:action LOAD-TRUCK
    :parameters    (?pkg - package ?truck - truck ?loc - place)
    :precondition  (and (at ?truck ?loc) (at ?pkg ?loc))
    :effect        (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))
                   ...)
```

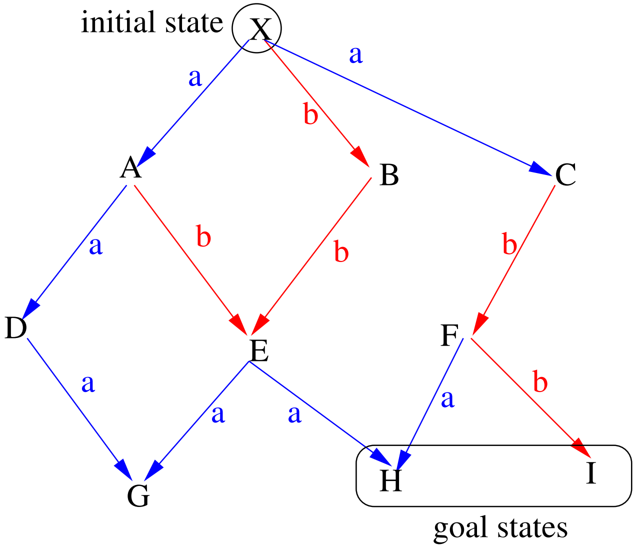
Lecture Overview

Planning Problems as Transition Systems

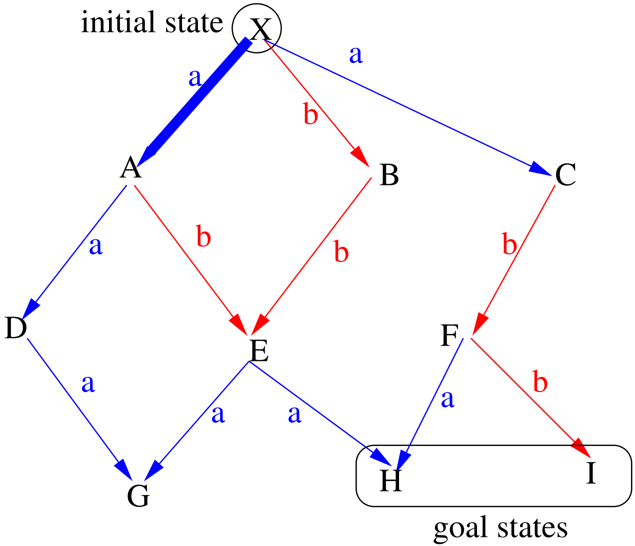
- We can view planning problems as searching for goal nodes in a large labeled graph (transition system)
- Nodes are defined by the value assignment to the fluents = states
- Labeled edges are defined by actions that change the appropriate fluents
- Use graph search techniques to find a (shortest) path in this graph!
- Note: The graph can become huge: 50 Boolean variables lead to $2^{50} = 10^{15}$ states

→ Create the transition system on the fly and visit only the parts that are *necessary*

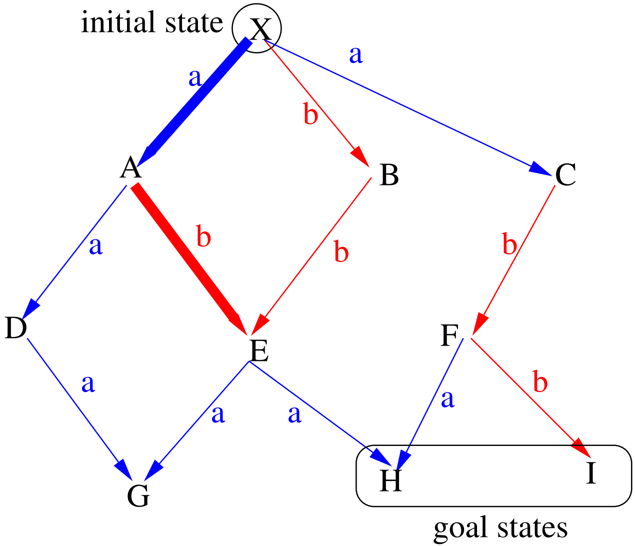
Transition System: Searching Through the State Space



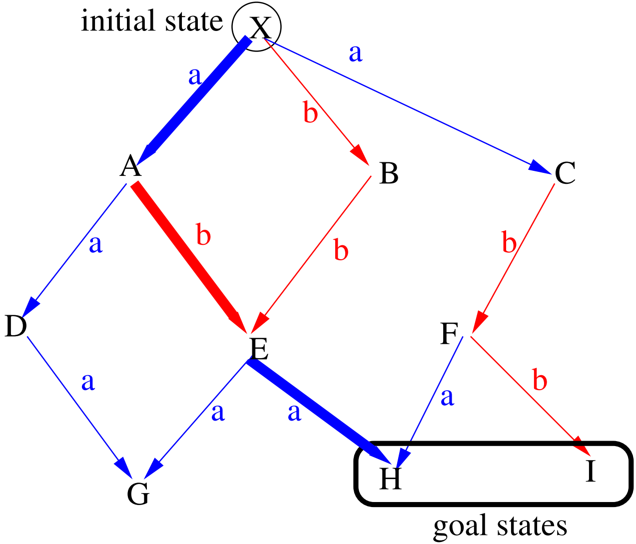
Transition System: Searching Through the State Space



Transition System: Searching Through the State Space



Transition System: Searching Through the State Space



Progression Planning: Forward Search

Search through transition system starting at initial state

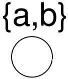
1. Initialize partial plan $\Delta := \langle \rangle$ and start at the unique initial state $\mathbf{I}$ and make it the current state S
2. Test whether we have reached a goal state already: $\mathbf{G} \subseteq S$? If so, return plan Δ.
3. Select one applicable action o_i non-deterministically and
   - compute successor state $S := App(S, o_i)$,
   - extend plan $\Delta := \langle \Delta, o_i \rangle$, and continue with step 2.

Instead of non-deterministic choice use some search strategy.
Progression planning can be easily extended to more expressive planning languages

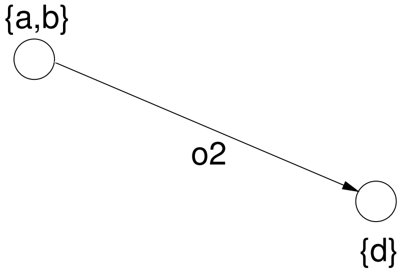
Not only boolean.

Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$

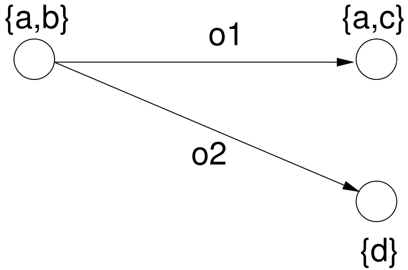


$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



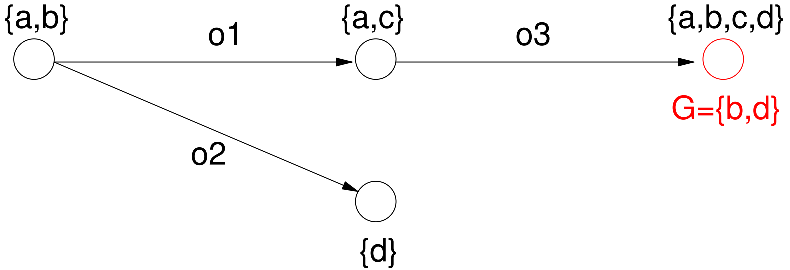
Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Regression Planning: Backward Search

Search through transition system starting at goal states. Consider sets of states, which are described by the atoms that are necessarily true in them

1. Initialize partial plan $\Delta := \langle\,\rangle$ and set $\mathbf{S} := \mathbf{G}$
2. Test whether we have reached the unique initial state already: $\mathbf{I} \supseteq \mathbf{S}$? If so, return plan Δ.
3. Select one action o_i non-deterministically which does not make (sub-)goals false ($\mathbf{S} \cap \neg \mathit{eff}^-(o_i) = \emptyset$) and
   - compute the regression of the description $\mathbf{S}$ through o_i:

     $$\mathbf{S} := \mathbf{S} - \mathit{eff}^+(o_i) \cup \mathit{pre}(o_i)$$

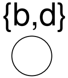
   - extend plan $\Delta := \langle o_i, \Delta \rangle$, and continue with step 2.

Instead of non-deterministic choice use some search strategy

Regression becomes much more complicated, if e.g. conditional effects are allowed. Then the result of a regression can be a general Boolean formula

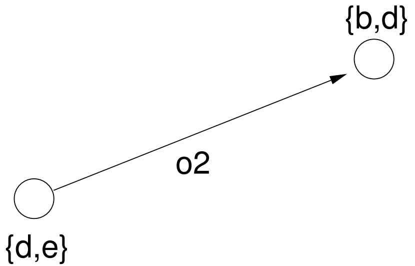
Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



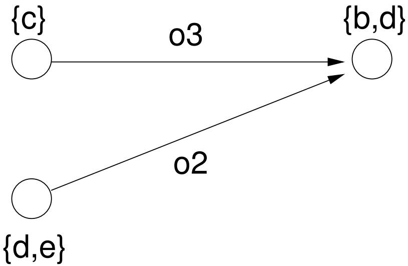
Regression Planning: Example

$$\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}$$



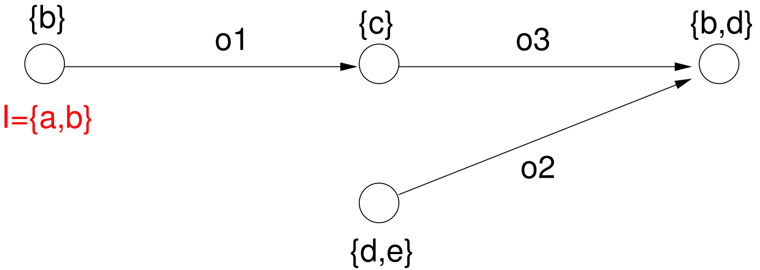
Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\}\rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\}\rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\}\rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Other Types of Search

- Of course, other types of search are possible.
- Change perspective: Do not consider the transition system as the space we have to explore, but consider the search through the space of (incomplete) plans:
  - Progression search: Search through the space of plan prefixes
  - Regression search: Search through plan suffixes
- Partial order planning:
  - Search through partially ordered plans by starting with the empty plan and trying to satisfy (sub-)goals by introducing new actions (or using old ones)
  - Make ordering choices only when necessary to resolve conflicts

Lecture Overview

The Planning Problem – Formally

Definition (Plan existence problem (PLANEX))

Instance: $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$.
Question: Does there exist a plan Δ that solves Π, i.e., $Res(\mathbf{I}, \Delta) \supseteq \mathbf{G}$?

Definition (Bounded plan existence problem (PLANLEN))

Instance: $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$ and a positive integer n.
Question: Does there exist a plan Δ of length n or less that solves Π?

From a practical point of view, also PLANGEN (*generating* a plan that solves Π) and PLANLENGEN (*generating* a plan of length n that solves Π) and PLANOPT (generating an optimal plan) are interesting (but at least as hard as the decision problems).

Basic STRIPS with First-Order Terms

- The state space for STRIPS with general first-order terms is **infinite**
- We can use function terms to describe (the index of) tape cells of a Turing machine
- We can use operators to describe the Turing machine control
- The existence of a plan is then equivalent to the existence of a successful computation on the Turing machine
- PLANEX for STRIPS with first-order terms can be used to decide the **Halting problem**

Theorem

PLANEX for STRIPS with first-order terms is ***undecidable****.*

Propositional STRIPS

Theorem

PLANEX is **PSPACE-complete** *for propositional STRIPS.*

→ Membership follows because we can successively guess operators and compute the resulting states (needs only polynomial space)

→ Hardness follows using again a generic reduction from TM acceptance. Instantiate polynomially many tape cells with no possibility to extend the tape (only poly. space, can all be generated in poly. time)

- PLANLEN is also PSPACE-complete (membership is easy, hardness follows by setting $k = 2^{|\Sigma|}$)

Restrictions on Plans

- If we restrict the length of the plans to be short, i.e., only **polynomial** in the size of the planning task, PLANEX becomes NP-complete
- Similarly, if we use a unary representation of the natural number k, then PLANLEN becomes NP-complete

$\rightarrow$ Membership obvious (guess & check)

$\rightarrow$ Hardness by a straightforward reduction from SAT or by a generic reduction.

- One source of complexity in planning stems from the fact that plans can become very long
- We are only interested in short plans!
- We can use methods for NP-complete problems if we are only looking for "short" plans.

Lecture Overview

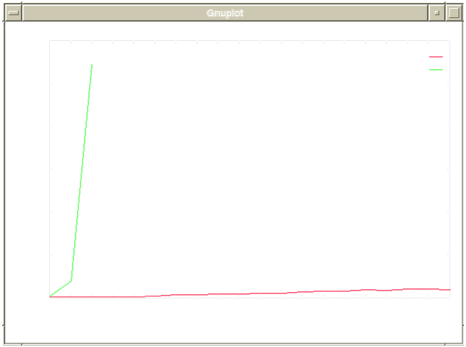
Current Approaches

- Planning as satisfiability: Iterative deepening.
- Planning with answer set programming.
- Symbolic planning with BDDs (finding many plans or non-det. plans)
- Heuristic forward-search planning (HSP, FF, FD)

Heuristic Search Planning

- Use an automatically generated heuristic estimator in order to select the next action or state
- Depending on the search scheme and the heuristic, the plan might not be the shortest one

→ It is often easier to go for sub-optimal solutions (remember *Logistics*)



Runtime of Heuristic search planner (red) vs. iterative deepening (green) on Gripper

Deriving Heuristics: Relaxations

- General principle for deriving heuristics:
  - Define a simplification (relaxation) of the problem and take the difficulty of a solution for the simplified problem as an heuristic estimator
- Example: straight-line distance on a map to estimate the travel distance
- Example: decomposition of a problem, where the components are solved ignoring the interactions between the components, which may incur additional costs
- In planning, one possibility is to ignore negative effects

Ignoring Negative Effects: Example

- In Logistics: The negative effects in *load* and *drive* are ignored:
- Simplified load operation: $load(O, V, P)$
  Precondition: $at(O, P), at(V, P), truck(V)$
  Effects: $\cancel{\neg at(O, P)}, in(O, V)$
- After loading, the package is still at the place and also inside the truck
- Simplified drive operation: $drive(V, P1, P2)$
  Precondition: $at(V, P1), truck(V), street(P1, P2)$
  Effects: $\cancel{\neg at(V, P1)}, at(V, P2)$
- After driving, the truck is in two places!

$\rightarrow$ We want the length of the shortest relaxed plan $\rightsquigarrow h^+(s)$

- How difficult is monotonic planning?

Lecture Overview

Current Trends in AI Planning

- Developing and analyzing heuristics
- Developing and anaylzing pruning techniques
- Develping new search techniques
- Extending the expressiveness of planning formalisms (and extending planning algorithms) in order to deal with
  - temporal planning,
  - planning with non-deterministic actions,
  - planning under partial observability,
  - planning with probabilistic effects,
  - multi-agent planning,
  - planning with epistemic goals,
- Reasoning about plans, e.g., diagnosing failures
- Judging morality of plans
- Applying/integrating planning technology
- Learning and planning
- …

Our Interests

- Foundation / theory
- Extending planning technology in order to cope with multi-agent scenarios and epistemic goals
- Using EVMDDs in modelling state-dependent costs
- Using planning techniques and extending them for robot control
- Using planning methodolgy in application in general
- Exploring the ethical dimension of planning systems

Lecture Overview

Summary

- Rational agents need to plan their course of action
- In order to describe planning tasks in a domain-independent, declarative way, one needs planning formalisms
- Basic STRIPS is a simple planning formalism, where actions are described by their preconditions in form of a conjunction of atoms and the effects are described by a list of literals that become true and false
- PDDL is the current “standard language” that has been developed in connection with the international planning competition
- Basic planning algorithms search through the space created by the transition system or through the plan space.
- Planning with STRIPS using first-order terms is undecidable
- Planning with propositional STRIPS is PSPACE-complete
- Since 1992, we have reasonably efficient planning method for propositional, classical STRIPS planning
- You can learn more about it in our planning class next term.