Plan-Space Search

• Searching for a Solution Plan in a Graph of Partial Plans
Literature

State-Space vs. Plan-Space Search

• state-space search: search through graph of nodes representing world states
  • search space directly corresponds to graph representation of state-transition system

• plan-space search: search through graph of partial plans
  • nodes: partially specified plans
  • arcs: plan refinement operations
  • least commitment principle: do not add constraints to the plan that are not strictly needed
  • solutions: partial-order plans
    • partial-order plan: set of actions + set of orderings; not necessarily total order
    • state-space algorithms also maintain partial plan – but always in total order
Overview

- The Search Space of Partial Plans
- Plan-Space Search Algorithms
- Extensions of the STRIPS Representation

Overview

The Search Space of Partial Plans

- now: defining the search space: partial plans + plan refinement operations

- Plan-Space Search Algorithms
- Extensions of the STRIPS Representation
Partial Plans

- plan: set of actions organized into some structure
- partial plan:
  - subset of the actions
  - subset of the organizational structure
    - temporal ordering of actions
    - rationale: what the action achieves in the plan
  - subset of variable bindings

Partial Plans

- plan: set of actions organized into some structure
  - organization e.g. sequence
- partial plan:
  - subset of the actions
  - subset of the organizational structure
    - temporal ordering of actions
    - rationale: what the action achieves in the plan
  - refers only to subset of actions
  - subset of variable bindings

- plan refinement operators accordingly: add actions, add ordering constraints, add causal links, add variable bindings
Adding Actions
• partial plan contains actions
  • initial state
  • goal conditions
  • set of operators with different variables

• reason for adding new actions
  • to achieve unsatisfied preconditions
  • to achieve unsatisfied goal conditions

Adding Actions
• partial plan contains actions
  • initial state
  • goal conditions
  • can be represented as two actions with only effects or preconditions
  • set of operators with different variables

• least commitment principle: introduce actions only for a reason
• reason for adding new actions
  • to achieve unsatisfied preconditions
  • to achieve unsatisfied goal conditions

• note: new actions can be added anywhere in the current partial plan
Adding Actions: Example

- empty plan:
  - initial state: all initially satisfied conditions (green)
  - goal: conditions that need to be satisfied (red)

- add operator: 1:move($r_1, l_1, m_1$)
  - number (1) to provide unique reference to this operator instance
  - also used as variable index for unique variables
  - least commitment principle: choose values for variables only when necessary

- add operator: 2:load($k_2, l_2, c_2, r_2$)
Adding Causal Links

- partial plan contains causal links
  - links from the provider
    - an effect of an action or
    - an atom that holds in the initial state
  - to the consumer
    - a precondition of an action or
    - a goal condition
- reasons for adding causal links
  - prevent interference with other actions

Adding Causal Links

- partial plan contains causal links
  - links from the provider
    - an effect of an action or
    - an atom that holds in the initial state
  - to the consumer
    - a precondition of an action or
    - a goal condition
  - causal link implies ordering constraint
    - but: provider need not come directly before consumer

- reasons for adding causal links
  - prevent interference with other actions
  - keeping track of rationale: any action inserted between provider and consumer must not clobber conditions in causal link
  - preconditions without a causal link pointing to them are open sub-gaols
Adding Causal Links: Example

• add link from 1:move to goal
  • changes colour of goal to green – now satisfied
• add link from 2:load to goal
• add link from initial state to 1:move
Adding Variable Bindings

- partial plan contains variable bindings
  - new operators introduce new (copies of) variables into the plan
  - solution plan must contain actions
  - variable binding constraints keep track of possible values for variables and co-designation
- reasons for adding variable bindings
  - to turn operators into actions
  - to unify and effect with the precondition it supports

Adding Variable Bindings

- partial plan contains variable bindings
  - new operators introduce new (copies of) variables into the plan
    - each copy of an operator has its own set of variables that are different from variables in other operators instances
  - solution plan must contain actions
  - variable binding constraints keep track of possible values for variables and co-designation
  - convention (here): give number to operator instances to distinguish them; let variables have index of operator they belong to
  - least commitment principle: add only necessary variable binding constraints
- reasons for adding variable bindings
  - to turn operators into actions
  - to unify and effect with the precondition it supports
Adding Variable Bindings: Example

- bind variables due to causal link:
  - bind $r_1$ to robot
  - bind $m_1$ to loc2
  - note: variables in operator no longer red to indicate they are bound
- clobbering: move may also destroy goal condition
- introduce variable inequality: $l_1 \neq \text{loc2}$
- clobbering now impossible
- introduce causal link from initial state
- bind $l_1$ to loc1
  - note consistency with inequality
Adding Ordering Constraints

• partial plan contains ordering constraints
  • binary relation specifying the temporal order between actions in the plan

• reasons for adding ordering constraints
  • all actions after initial state
  • all actions before goal
  • causal link implies ordering constraint
  • to avoid possible interference

• interference can be avoided by ordering the potentially interfering action before the provider or after the consumer of a causal link

• least commitment principle: introduce ordering constraints only if necessary

• result: solution plan not necessarily totally ordered
Adding Ordering Constraints: Example

- ordering constraints
  - due to causal links
  - also: all actions before goal
- ordering: all actions after initial state
- orderings may occur between actions
Definition of Partial Plans

- A partial plan is a tuple $\pi = (A, <, B, L)$, where:
  - $A = \{a_1, \ldots, a_k\}$ is a set of partially instantiated planning operators;
  - $<$ is a set of ordering constraints on $A$ of the form $(a_i < a_j)$;
  - $B$ is a set of binding constraints on the variables of actions in $A$ of the form $x = y$, $x \neq y$, or $x \in D_x$;
  - $L$ is a set of causal links of the form $\langle a_i \leftarrow [p] \rightarrow a_j \rangle$ such that:
    - $a_i$ and $a_j$ are actions in $A$;
    - the constraint $(a_i < a_j)$ is in $<$;
    - proposition $p$ is an effect of $a_i$ and a precondition of $a_j$; and
    - the binding constraints for variables in $a_i$ and $a_j$ appearing in $p$ are in $B$.

- sub-gaols in a partial plan: preconditions without causal links
- different view: partial plan as set of (sequential) plans
  - those that meet the specified constraints and can be refined to a total order plan by adding constraints
- note: partial plans with two types of additional flexibility:
  - actions only partially ordered and
  - not all variables need to be instantiated
Plan-Space Search: Initial Search State

- represent initial state and goal as dummy actions
  - init: no preconditions, initial state as effects
  - goal: goal conditions as preconditions, no effects
- empty plan $\pi_0 = (\{\text{init, goal}\}, \{(\text{init} \prec \text{goal})\}, \{\}, \{\})$:
  - two dummy actions init and goal;
  - one ordering constraint: init before goal;
  - no variable bindings; and
  - no causal links.

Plan-Space Search: Initial Search State

- problem: plan space representation does not maintain states, but need to give initial state and goal description
  - represent initial state and goal as dummy actions
    - init: no preconditions, initial state as effects
    - goal: goal conditions as preconditions, no effects
  - empty plan $\pi_0 = (\{\text{init, goal}\}, \{(\text{init} < \text{goal})\}, \{\}, \{\})$:
    - two dummy actions init and goal;
    - one ordering constraint: init before goal;
    - no variable bindings; and
    - no causal links.
Plan-Space Search: Initial Search State Example

• note empty box for preconditions in init and empty box for effects in goal
Plan-Space Search: Successor Function

- states are partial plans
- generate successor through plan refinement operators (one or more):
  - adding an action to $A$
  - adding an ordering constraint to $≺$
  - adding a binding constraint to $B$
  - adding a causal link to $L$

• more required to keep partial plans consistent, e.g. adding a causal link implies adding an ordering constraint

• adding an action to $A$
• adding an ordering constraint to $≺$
• adding a binding constraint to $B$
• adding a causal link to $L$

• successors must be consistent: constraints in a partial plan must be satisfiable

• plan-space planning decouple two sub-problems:
  • which actions need to be performed
  • how to organize these actions

• partial plan as set of plans: refinement operation reduces the set to smaller subset

• next: to define planning as plan-space search problem: need to define goal state
Total vs. Partial Order

Let $\mathcal{P} = (\Sigma, \textit{s}, g)$ be a planning problem. A plan $\pi$ is a solution for $\mathcal{P}$ if $\gamma(\textit{s}, \pi)$ satisfies $g$.

- Problem: $\gamma(\textit{s}, \pi)$ only defined for sequence of ground actions
  - Partial order corresponds to total order in which all partial order constraints are respected
  - Partial instantiation corresponds to grounding in which variables are assigned values consistent with binding constraints

Total vs. Partial Order

Let $\mathcal{P} = (\Sigma, \textit{s}, g)$ be a planning problem. A plan $\pi$ is a solution for $\mathcal{P}$ if $\gamma(\textit{s}, \pi)$ satisfies $g$.

- Solution defined for state transition system
- Problem: $\gamma(\textit{s}, \pi)$ only defined for sequence of ground actions
  - Partial order corresponds to total order in which all partial order constraints are respected
    - Partial ordering is consistent iff it is free of loops
    - Note: there may be an exponential number of total ordering consistent with a given partial ordering
  - Partial instantiation corresponds to grounding in which variables are assigned values consistent with binding constraints
    - Note: exponential combinatorics of assigning values to variables
Partial Order Solutions

Let $\mathcal{P}=(\Sigma,s_i,g)$ be a planning problem. A plan $\pi = (A,\prec,B,L)$ is a (partial order) solution for $\mathcal{P}$ if:

- its ordering constraints $\prec$ and binding constraints $B$ are consistent; and

- for every sequence $\langle a_1,...,a_k \rangle$ of all the actions in $A-\{\text{init}, \text{goal}\}$ that is
  - totally ordered and grounded and respects $\prec$ and $B$
  - $\gamma(s_i, \langle a_1,...,a_k \rangle)$ must satisfy $g$.

- note: causal links do not play a role in the definition of a solution

- with exponential number of sequences to check, definition is not very useful (as computational procedure for goal test)

- idea: use causal links to verify that every precondition of every action is supported by some other action

- problem: condition not strong enough
**Threat: Example**

- start with partial plan from previous example (grounded; initial state not shown due to limited space on slide)
- introduce new 3:move action to achieve at(robot,loc1) precondition of 2:load action
  - note: still many unachieved preconditions – not a solution yet
- add causal link to maintain rationale
- add ordering to be consistent with causal link
- new: label causal link with condition it protects
- threat: effect of 1:move is negation of condition protected by causal link
  - if 1:move is executed between 3:move and 2:load the plan is no longer valid
- possible solution: additional ordering constraint
Threats

• An action $a_k$ in a partial plan $\pi = (A, \prec, B, L)$ is a threat to a causal link $\langle a_i, \neg[p] \rightarrow a_j \rangle$ iff:
  • $a_k$ has an effect $\neg q$ that is possibly inconsistent with $p$, i.e. $q$ and $p$ are unifiable;
  • the ordering constraints ($a_i \prec a_k$) and ($a_k \prec a_j$) are consistent with $\prec$; and
  • the binding constraints for the unification of $q$ and $p$ are consistent with $B$. 
Flaws

• A flaw in a plan $\pi = (A,\prec,B,L)$ is either:
  • an unsatisfied sub-goal, i.e. a precondition of an action in $A$ without a causal link that supports it; or
  • a threat, i.e. an action that may interfere with a causal link.

Flaws

• A flaw in a plan $\pi = (A,\prec,B,L)$ is either:
  • an unsatisfied sub-goal, i.e. a precondition of an action in $A$ without a causal link that supports it; or
  • a threat, i.e. an action that may interfere with a causal link.
Flawless Plans and Solutions

• Proposition: A partial plan \( \pi = (A, <, B, L) \) is a solution to the planning problem \( P = (\Sigma, s_i, g) \) if:
  • \( \pi \) has no flaw;
  • the ordering constraints < are not circular; and
  • the variable bindings \( B \) are consistent.
• Proof: by induction on number of actions in \( A \)
  • base case: empty plan
  • induction step: totally ordered plan minus first step is solution implies plan including first step is a solution:
    \( \gamma(s_i, \langle a_1, \ldots, a_k \rangle) = \gamma(\gamma(s_i, a_1), \langle a_2, \ldots, a_k \rangle) \)

Flawless Plans and Solutions

• Proposition: A partial plan \( \pi = (A, <, B, L) \) is a solution to the planning problem \( P = (\Sigma, s_i, g) \) if:
  • \( \pi \) has no flaw;
  • the ordering constraints < are not circular; and
  • the variable bindings \( B \) are consistent.

• computation:
  • let partial plans in the search space only violate the first condition (have flaws)
  • partial plans that violate either of the last two conditions cannot be refined into a solution and need not be generated

• Proof: by induction on number of actions in \( A \)
  • base case: empty plan
    • no flaws – every goal condition is supported by causal link from initial state
  • induction step: totally ordered plan minus first step is solution implies plan including first step is a solution:
    \( \gamma(s_i, \langle a_1, \ldots, a_k \rangle) = \gamma(\gamma(s_i, a_1), \langle a_2, \ldots, a_k \rangle) \)
    • truncated plan is solution to different problem
Overview

- The Search Space of Partial Plans
  - just done: defining the search space: partial plans + plan refinement operations

- Plan-Space Search Algorithms
  - now: an algorithm that performs the search through the space of partial plans

- Extensions of the STRIPS Representation
Plan-Space Planning as a Search Problem

- given: statement of a planning problem $P=(O,s,g)$
- define the search problem as follows:
  - initial state: $\pi_0 = \{\text{init, goal}\},\{\text{init<goal}\},\{\},\{\}$
  - goal test for plan state $p$: $p$ has no flaws
  - path cost function for plan $\pi$: $|\pi|$
  - successor function for plan state $p$: refinements of $p$ that maintain $<$ and $B$

Plan-Space Planning as a Search Problem

- given: statement of a planning problem $P=(O,s,g)$
- define the search problem as follows:
  - initial state: $\pi_0 = \{\text{init, goal}\},\{\text{init<goal}\},\{\},\{\}$
  - goal test for plan state $p$: $p$ has no flaws
  - path cost function for plan $\pi$: $|\pi|$
  - successor function for plan state $p$: refinements of $p$ that maintain $<$ and $B$

- note: plan space may be infinite even when state space is finite
PSP Procedure: Basic Operations

• PSP: Plan-Space Planner
• main principle: refine partial π plan while maintaining < and B consistent until π has no more flaws
• basic operations:
  • find the flaws of π, i.e. its sub-goals and its threats
  • select one of the flaws
  • find ways to resolve the chosen flaw
  • choose one of the resolvers for the flaw
  • refine π according to the chosen resolver

PSP Procedure: Basic Operations

• find the flaws of π, i.e. its sub-goals and its threats
  • simple for empty plan – all goal conditions are unachieved sub-goals and no threats
• select one of the flaws
• find ways to resolve the chosen flaw
• choose one of the resolvers for the flaw
• refine π according to the chosen resolver

• modify the plan in such a way that < and B are in a consistent state for the generated successor
  • aim: no need to verify consistency of < and B for goal test
• **PSP: Pseudo Code**

  • **function** PSP(\( plan \))
    - refines the given partial plan into a solution plan; start with initial plan \( \pi_0 \)

  • \( allFlaws \leftarrow plan.openGoals() + plan.threats() \)

  • **if** \( allFlaws.empty() \) **then return** \( plan \)
    - see proposition in previous section: no flaws implies solution

  • \( flaw \leftarrow allFlaws.selectOne() \)

  • \( allResolvers \leftarrow flaw.getResolvers(plan) \)
    - represents all possible ways of removing the selected flaw from the partial plan

  • **if** \( allResolvers.empty() \) **then return** failure
    - no resolvers means plan cannot be made flawless

  • \( resolver \leftarrow allResolvers.chooseOne() \)

  • \( newPlan \leftarrow plan.refine(resolver) \)
    - must maintain consistency of \( \prec \) and \( B \); new plan may contain new flaws

  • **return** PSP(\( newPlan \))
PSP: Choice Points

- `resolver ← allResolvers.chooseOne()`
  - non-deterministic choice
- `flaw ← allFlaws.selectOne()`
  - deterministic selection
  - all flaws need to be resolved before a plan becomes a solution
  - order not important for completeness
  - order is important for efficiency

- for finding first plan, not so for finding all plans
- deterministic implementation: using IDA*, for example
Implementing *plan.openGoals()*

- finding unachieved sub-goals (incrementally):
  - in $\pi_0$: goal conditions
  - when adding an action: all preconditions are unachieved sub-goals
  - when adding a causal link: protected proposition is no longer unachieved

Implementing *plan.openGoals()*

- finding unachieved sub-goals (incrementally):
  - in $\pi_0$: goal conditions
  - when adding an action: all preconditions are unachieved sub-goals
  - when adding a causal link: protected proposition is no longer unachieved
Implementing \texttt{plan.threats()}

- finding threats (incrementally):
  - in π₀: no threats
  - when adding an action $a_{\text{new}}$ to $\pi = (A,≺,B,L)$:
    - for every causal link $\langle a_i - [p] \rightarrow a_j \rangle \in L$
      - if ($a_{\text{new}} < a_i$) or ($a_j < a_{\text{new}}$) then next link
      - else for every effect $q$ of $a_{\text{new}}$
        - if ($\exists \sigma$: $\sigma(p) = \sigma(\neg q)$) then $q$ of $a_{\text{new}}$ threatens $\langle a_i - [p] \rightarrow a_j \rangle$
  - when adding a causal link $\langle a_i - [p] \rightarrow a_j \rangle$ to $\pi = (A,≺,B,L)$:
    - for every action $a_{\text{old}} \in A$
      - if ($a_{\text{old}} < a_i$) or ($a_j = a_{\text{old}}$) or ($a_j < a_{\text{old}}$) then next action
      - else for every effect $q$ of $a_{\text{old}}$
        - if ($\exists \sigma$: $\sigma(p) = \sigma(\neg q)$) then $q$ of $a_{\text{old}}$ threatens $\langle a_i - [p] \rightarrow a_j \rangle$
Implementing `flaw.getResolvers(plan)`

- for unachieved precondition $p$ of $a_g$:
  - add causal links to an existing action:
    - for every action $a_{old} \in A$
      - if $(a_g = a_{old})$ or $(a_g < a_{old})$ then next action
    - else for every effect $q$ of $a_{old}$
      - if $(\exists \sigma: \sigma(p) = \sigma(q))$ then adding
        $\langle a_{old} - [\sigma(p)] \rightarrow a_g \rangle$ is a resolver
  - add a new action and a causal link:
    - for every effect $q$ of every operator $o$
      - if $(\exists \sigma: \sigma(p) = \sigma(q))$ then adding
        $a_{new} = o.newInstance()$ and
        $\langle a_{new} - [\sigma(p)] \rightarrow a_g \rangle$ is a resolver
Implementing `flaw.getResolvers(plan)`

- for effect `q` of action `a_i` threatening `<a_i-[p]→ a_j>`:
  - order action before threatened link:
    - if `(a_i=a_j)` or `(a_j<a_i)` then not a resolver
    - else adding `(a_i→a_j)` is a resolver
  - order threatened link before action:
    - if `(a_i=a_j)` or `(a_i<q)` then not a resolver
    - else adding `(a_i<a_j)` is a resolver
  - extend variable bindings such that unification fails:
    - for every variable `v` in `p` or `q`
      - if `v≠σ(v)` is consistent with `B`
      - adding `v≠σ(v)` is a resolver
Implementing `plan.refine(resolver)`

- refines partial plan with elements in resolver by adding:
  - an ordering constraint;
  - one or more binding constraints;
  - a causal link; and/or
  - a new action.
- no testing required
- must update flaws:
  - unachieved preconditions (see: `plan.openGoals()`)
  - threats (see: `plan.threats()`)

Implementing `plan.refine(resolver)`

- refines partial plan with elements in resolver by adding:
  - an ordering constraint;
  - one or more binding constraints;
  - a causal link; and/or
  - a new action.
- no testing required
  - all testing already done in `flaw.getResolvers(plan)`
- must update flaws:
  - unachieved preconditions (see: `plan.openGoals()`)
  - threats (see: `plan.threats()`)


Maintaining Ordering Constraints

• required operations:
  • query whether \((a_i \prec a_j)\)
  • adding \((a_i \prec a_j)\)

• possible internal representations:
  • maintain set of predecessors/successors for each action as given
  • maintain only direct predecessors/successors for each action
  • maintain transitive closure of \(\prec\) relation

• without consistency testing

• operations have different time and space complexity

• note: query performed more often than addition
Maintaining Variable Binding Constraints

- types of constraints:
  - unary constraints: \( x \in D_x \)
  - equality constraints: \( x = y \)
  - inequalities: \( x \neq y \)

- note: general CSP problem is NP-complete
PSP: Data Flow

• deterministic step: selecting a flaw
  • no backtracking required
  • selection important for efficiency
  • heuristic guidance required

• non-deterministic step: choosing a resolver for a flaw
  • implemented as backtracking
    • order in which resolvers are tried important for efficiency
    • heuristic guidance required

• note: admissible heuristics (A*) must have step cost greater than zero
PSP: Sound and Complete

• Proposition: The PSP procedure is sound and complete: whenever $\pi_0$ can be refined into a solution plan, $\text{PSP}(\pi_0)$ returns such a plan.

• Proof:
  • soundness: $\prec$ and $B$ are consistent at every stage of the refinement
  • completeness: induction on the number of actions in the solution plan

• note: non-deterministic version is complete, deterministic implementation must avoid infinite branches
PSP Implementation: PoP

- based on UCPOP

- **extended input:**
  - partial plan (as before)
  - agenda: set of pairs \((a,p)\) where \(a\) is an action and \(p\) is one of its preconditions

- search control by flaw type
  - unachieved sub-goal (on agenda): as before
  - threats: resolved as part of the successor generation process

PSP Implementation: PoP

- based on UCPOP

- **extended input:**
  - partial plan (as before)
  - agenda: set of pairs \((a,p)\) where \(a\) is an action and \(p\) is one of its preconditions
  - initial agenda: one pair for each precondition of the goal step

- search control by flaw type
  - unachieved sub-goal (on agenda): as before
  - threats: resolved as part of the successor generation process
PoP: Pseudo Code (1)

function PoP(plan, agenda)
if agenda.empty() then return plan
(ag_p, pg) ← agenda.selectOne()
agenda ← agenda - (ag_p, pg)
relevant ← plan.getProviders(pg)
if relevant.empty() then return failure
(ap_p, pp, σ) ← relevant.chooseOne()
plan.L ← plan.L ∪ \langle ap_p - [p] \rightarrow ag_p \rangle
plan.B ← plan.B ∪ σ

PoP: Pseudo Code (1)
• function PoP(plan, agenda)
• if agenda.empty() then return plan
• (ag_p, pg) ← agenda.selectOne()
  • deterministic choice point
• agenda ← agenda - (ag_p, pg)
• relevant ← plan.getProviders(pg)
  • finds all actions
    • either from within the plan or
    • from new instances of an operator
      • that have an effect that unifies with condition
• if relevant.empty() then return failure
• (ap_p, pp, σ) ← relevant.chooseOne()
  • non-deterministic choice point
• plan.L ← plan.L ∪ \langle ap_p - [p] \rightarrow ag_p \rangle
• plan.B ← plan.B ∪ σ
  • must succeed for elements of relevant
PoP: Pseudo Code (2)

• if \( a_p \notin \text{plan.A} \) then
  • if the action is new and needs to be added to the plan
  • \( \text{plan.add}(a_p) \)
    • involves updating set of actions and ordering constraints
  • \( \text{agenda} \leftarrow \text{agenda} + a_p.\text{preconditions} \)
    • all preconditions of the new action are new sub-goals
  • \( \text{newPlan} \leftarrow \text{plan} \)
  • for each \( \text{threat on } \langle a_p - [p] \rightarrow a_g \rangle \) or due to \( a_p \) do
    • note: two sources of threats are treated identically
    • \( \text{allResolvers} \leftarrow \text{threat.getResolvers(newPlan)} \)
  • if \( \text{allResolvers}.\text{empty()} \) then return failure
  • \( \text{resolver} \leftarrow \text{allResolvers}.\text{chooseOne()} \)
  • second non-deterministic choice point
  • \( \text{newPlan} \leftarrow \text{newPlan}.\text{refine(resolver)} \)
    • note: loop does not add to agenda
  • return \( \text{PSP(newPlan,agenda)} \)
State-Space vs. Plan-Space Planning

- **state-space planning**
  - finite search space
  - explicit representation of intermediate states
  - action ordering reflects control strategy
  - causal structure only implicit
  - search nodes relatively simple and successors easy to compute

- **plan-space planning**
  - finite search space
  - no intermediate states
  - choice of actions and organization independent
  - explicit representation of rationale
  - search nodes are complex and successors expensive to compute

---

State-Space vs. Plan-Space Planning

- **state-space planning vs. plan-space planning**
  - finite search space vs. finite search space
    - important: portion of search space explored/generated
  - explicit representation of intermediate states vs. no intermediate states
    - explicit representation allows for efficient domain specific heuristics and control knowledge
  - action ordering reflects control strategy vs. choice of actions and organization independent
  - causal structure only implicit vs. explicit representation of rationale
    - important for plan execution
  - search nodes relatively simple and successors easy to compute vs. search nodes are complex and successors expensive to compute
Using Partial-Order Plans: Main Advantages

• more flexible during execution
• using constraint managers facilitates extensions such as:
  • temporal constraints
  • resource constraints
• distributed and multi-agent planning fit naturally into the framework

Using Partial-Order Plans: Main Advantages

• more flexible during execution
  • saved rationale facilitates execution monitoring and re-planning
• using constraint managers facilitates extensions such as:
  • temporal constraints
  • resource constraints
  • both very important for realistic planning
• distributed and multi-agent planning fit naturally into the framework
Overview

- The Search Space of Partial Plans
- Plan-Space Search Algorithms
  - just done: an algorithm that performs the search through the space of partial plans
- Extensions of the STRIPS Representation
  - now: extensions to the restricted STRIPS representation and approaches to deal with them
Existential Quantification in Goals

- allow existentially quantified conjunction of literals as goal:
  - \( g = \exists x_1, \ldots, x_n : l_1 \land \ldots \land l_m \)
- rewrite into equivalent planning problem:
  - new goal \( g' = \{ p \} \) where \( p \) is an unused proposition symbol
  - introduce additional operator \( o = (\text{op-g}(x_1, \ldots, x_n), \{l_1, \ldots, l_m\}, \{p\}) \)
- in plan-space search: no change needed

**Existential Quantification in Goals**

- allow existentially quantified conjunction of literals as goal:
  - \( g = \exists x_1, \ldots, x_n : l_1 \land \ldots \land l_m \)
  - but still no free variables in the goal literals
- rewrite into equivalent planning problem:
  - new goal \( g' = \{ p \} \) where \( p \) is an unused proposition symbol
  - introduce additional operator \( o = (\text{op-g}(x_1, \ldots, x_n), \{l_1, \ldots, l_m\}, \{p\}) \)
  - solution plans will be one step longer, with last step being an instantiation of op-g
- no increase in expressive power (can rewrite every problem), but sometimes makes representation appear more natural
- in plan-space search: no change needed
  - partial plans do not require all variables to be instantiated
Example: Existential Quantification in Goals

• goal: $\exists x, y: \text{on}(x,c1) \land \text{on}(y,c2)$
• rewritten goal: $p$
• new operator: $o = (\text{op-g}(x,y),\{\text{on}(x,c1),\text{on}(y,c2)\},\{p\})$
• plan-space search goal: $\text{on}(x,c1) \land \text{on}(y,c2)$

• variables in goals are implicitly existentially quantified
Typed Variables

- allow typed variables in operators:
  - name(o) = n(x_1:t_1,…,x_k:t_k) where t_i is the type of variable x_i

- rewrite into equivalent planning problem:
  - add preconditions {t_1(x_1),…,t_k(x_k)} to o
  - if constant c_i is of type t_j, add t_j(c_i) to the initial state
  - remove types from operator names

Typed Variables

- allow typed variables in operators:
  - name(o) = n(x_1:t_1,…,x_k:t_k) where t_i is the type of variable x_i
    - types usually given with problem specification
    - exact syntax for typing not important

- rewrite into equivalent planning problem:
  - add preconditions {t_1(x_1),…,t_k(x_k)} to o
    - types are unary predicates here
  - if constant c_i is of type t_j, add t_j(c_i) to the initial state
  - remove types from operator names

- similarly: typed relations

- advantages: readability, reduced number of actions (operator instances)
DWR Example: Typed Variables

• operator: move(r:robot,l:location,m:location)
  • precond: adjacent(l,m), at(r,l), ¬occupied(m)
  • effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r,l)

• rewritten operator: move(r,l,m)
  • precond: adjacent(l,m), at(r,l), ¬occupied(m), robot(r), location(l), location(m)
  • effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r,l)

• rewritten initial state:
  • s_i = {robot(r1), container(c1), container(c2),…}

• note: dealing with typed variables directly in the planner is far more efficient
Conditional Operators

- Conditional planning operators:
  - \( o = (n,(precond_0,effects_0),..., (precond_n, effects_n)) \)
    - \( n = o(x_1, ..., x_n) \) as before,
    - \( (precond_0, effects_0) \) are the unconditional preconditions and effects of the operator, and
    - \( (precond_i, effects_i) \) for \( i \geq 1 \) are the conditional preconditions and effects of the operator.
  - A ground instance \( a \) of \( o \) is applicable in state \( s \) if \( s \) satisfies \( precond_0 \)
  - Let \( \{i \in [0,n] \mid s \text{ satisfies } precond_i(a)\} \); then:
    \( \gamma(s,a) = (s - \bigcup_{i \in I} effects^{-}(a)) \cup (\bigcup_{i \in I} effects^{+}(a)) \)
DWR Example: Conditional Operators

- relation $\text{at}(o,l)$: object $o$ is at location $l$
- conditional move operator:
  $\text{move}(r,l,m,c)$
  - $\text{precond}_0$: $\text{adjacent}(l,m)$, $\text{at}(r,l)$, $\neg \text{occupied}(m)$
  - $\text{effects}_0$: $\text{at}(r,m)$, $\text{occupied}(m)$, $\neg \text{occupied}(l)$, $\neg \text{at}(r,l)$
  - $\text{precond}_1$: $\text{loaded}(r,c)$
  - $\text{effects}_1$: $\text{at}(c,m)$, $\neg \text{at}(c,l)$

DWR Example: Conditional Operators

- relation $\text{at}(o,l)$: object $o$ is at location $l$
  - at previously only used for location of robot
- conditional move operator:
  $\text{move}(r,l,m,c)$
  - new parameter $c$, the potentially loaded container
  - $\text{precond}_0$: $\text{adjacent}(l,m)$, $\text{at}(r,l)$, $\neg \text{occupied}(m)$
  - $\text{effects}_0$: $\text{at}(r,m)$, $\text{occupied}(m)$, $\neg \text{occupied}(l)$, $\neg \text{at}(r,l)$
    - as before
  - $\text{precond}_1$: $\text{loaded}(r,c)$
  - $\text{effects}_1$: $\text{at}(c,m)$, $\neg \text{at}(c,l)$

  - if the container is loaded it will move with the robot
Extending PoP to handle Conditional Operators

- modifying `plan.getProviders(pg)`:
  - new action with matching conditional effect
  - add precondition of conditional effect to agenda
- managing conditional threats:
  - new alternative resolver: add negated precondition of threatening conditional effect to agenda

Extending PoP to handle Conditional Operators

- modifying `plan.getProviders(pg)`:
  - new action with matching conditional effect
  - add precondition of conditional effect to agenda
    - along with unconditional preconditions
- managing conditional threats:
  - when the threatening effect is a conditional effect
    - new alternative resolver: add negated precondition of threatening conditional effect to agenda
    - other alternatives: ordering and binding constraints
Quantified Expressions

- allow universally quantified variables in conditional preconditions and effects:
  - for-all $x_1,\ldots,x_n$: (precond$_i$,effects$_i$)
- $a$ is applicable in state $s$ if $s$ satisfies precond$_0$
- Let $\sigma$ be a substitution for $x_1,\ldots,x_n$ such that $\sigma$(precond$_i$(a)) and $\sigma$(effects$_i$(a)) are ground.
  - If $s$ satisfies $\sigma$(precond$_i$(a)) then
  - $\sigma$(effects$_i$(a)) are effects of the action.

Quantified Expressions

- allow universally quantified variables in conditional preconditions and effects:
  - for-all $x_1,\ldots,x_n$: (precond$_i$,effects$_i$)
- $a$ is applicable in state $s$ if $s$ satisfies precond$_0$
  - applicability depends only on unconditional preconditions
- Let $\sigma$ be a substitution for $x_1,\ldots,x_n$ such that $\sigma$(precond$_i$(a)) and $\sigma$(effects$_i$(a)) are ground.
  - If $s$ satisfies $\sigma$(precond$_i$(a)) then
  - $\sigma$(effects$_i$(a)) are effects of the action.
  - conditional effect occurs for all possible substitutions where the state satisfies the conditional preconditions
- note: cannot use binding constraints to resolve conditional threats here
DWR Example: Quantified Expressions

- extension: robots can carry multiple containers
- extended move operator: \(\text{move}(r,l,m)\)
  - \(\text{precond}_0\): \text{adjacent}(l,m), \text{at}(r,l), \neg \text{occupied}(m)
  - \(\text{effects}_0\): \text{at}(r,m), \text{occupied}(m), \neg \text{occupied}(l), \neg \text{at}(r,l)
  - for-all \(x\):
    - \(\text{precond}_1\): \text{loaded}(r,x)
    - \(\text{effects}_1\): \text{at}(x,m), \neg \text{at}(x,l))
Disjunctive Preconditions

• allow alternatives (disjunctions) in preconditions:
  • precond = precond₁ v…v precondₙ
  • a is applicable in state s if s satisfies at least one of precond₁ … precondₙ
  • effects remain unchanged

• rewrite:
  • replace operator with n disjunctive preconditions by n operators with precondᵢ as precondition

• leads to exponentially larger search space; more efficiently handled in the planner
DWR Example: Disjunctive Preconditions

• robot can move between locations if there is a road between them or the robot has all-wheel drive

• extended move operator:
  \[
  \text{move}(r,l,m)
  \]
  \[
  \begin{align*}
  \text{precond: } & (\text{road}(l,m), \text{at}(r,l), \neg \text{occupied}(m)) \lor \text{all-wheel-drive}(r), \text{at}(r,l), \neg \text{occupied}(m) \\
  \text{effects: } & \text{at}(r,m), \text{occupied}(m), \neg \text{occupied}(l), \neg \text{at}(r,l)
  \end{align*}
  \]

• more complex formulae can be transformed into DNF

• effects: \text{at}(r,m), \text{occupied}(m), \neg \text{occupied}(l), \neg \text{at}(r,l)
Axiomatic Inference: Static Case

• axioms over rigid relations:
  • example: \( \forall l_1, l_2: \text{adjacent}(l_1, l_2) \leftrightarrow \text{adjacent}(l_2, l_1) \)

• state-specific axioms:
  • example: \( \forall c: \text{container}(c) \leftrightarrow \text{at}(c, \text{loc1}) \) holds in \( s_i \)

• approach: pre-compute

Axiomatic Inference: Static Case

• idea: represent knowledge that is not explicit in the state of the world; derived knowledge

• axioms over rigid relations:
  • example: \( \forall l_1, l_2: \text{adjacent}(l_1, l_2) \leftrightarrow \text{adjacent}(l_2, l_1) \)
  • adjacent is a symmetric relationship

• state-specific axioms:
  • example: \( \forall c: \text{container}(c) \leftrightarrow \text{at}(c, \text{loc1}) \) holds in \( s_i \)
  • in the initial state, all containers are at location \( \text{loc1} \)

• approach: pre-compute
  • rigid relations: cannot appear in effects, truth value does not change from state to state; hence, can be pre-computed
  • state-specific axioms: expansion into ground atoms possible because of finite domains
    • same technique for quantified effects
Axiomatic Inference: Dynamic Case

- axioms over flexible relations:
  - example: $\forall k, x: \neg\text{holding}(k, x) \leftrightarrow \text{empty}(k)$
  - approach:
    - divide relations into primary and secondary where secondary relations do not appear in effects
    - transform axioms into implications where primary relations must not appear in right-hand side
  - example:
    - primary: holding / secondary: empty
      - $\forall k \neg \exists x: \text{holding}(k, x) \rightarrow \text{empty}(k)$
      - $\forall k \exists x: \text{holding}(k, x) \rightarrow \neg\text{empty}(k)$

Axiomatic Inference: Dynamic Case

- axioms over flexible relations:
  - example: $\forall k, x: \neg\text{holding}(k, x) \leftrightarrow \text{empty}(k)$
  - a crane is empty iff it is not holding anything
  - approach:
    - divide relations into primary and secondary where secondary relations do not appear in effects
      - primary relations may appear in preconditions and effects, secondary relations only in preconditions
    - transform axioms into implications where primary relations must not appear in right-hand side
      - truth value of secondary relations depends on primary relations which are modified by operators
  - example:
    - primary: holding / secondary: empty
      - must remove empty-relation from effects of all operators
      - $\forall k \neg \exists x: \text{holding}(k, x) \rightarrow \text{empty}(k)$
      - $\forall k \exists x: \text{holding}(k, x) \rightarrow \neg\text{empty}(k)$
Extended Goals

• not part of classical planning formalisms

• some problems can be translated into equivalent classical problems, e.g.
  • states to be avoided: add corresponding preconditions to operators
  • states to be visited twice: introduce visited relation and maintain in operators
  • constraints on solution length: introduce count relation that is increased with each step

• requires function symbols and integer arithmetic (attached procedures)
Other Extensions

• Function Symbols
  • infinite domains, undecidable in general

• Attached Procedures
  • evaluate relations using special code rather than general inference
    • efficiency may be necessary in real-world domains
    • variables must usually be bound to evaluate relations
    • semantics of such relations

• expansion into ground atoms no longer possible, reasoning within one state is undecidable

• evaluate relations using special code rather than general inference
  • efficiency may be necessary in real-world domains
    • example: numeric computations
  • variables must usually be bound to evaluate relations
    • difficult with least commitment approach taken by many planners
  • semantics of such relations
Overview

- The Search Space of Partial Plans
- Plan-Space Search Algorithms
- Extensions of the STRIPS Representation

- just done: extensions to the restricted STRIPS representation and approaches to deal with them