Planning, Execution & Learning

1. Transformational Planning

Reid Simmons
Transformational Planning

• Basic Idea
  – *Create new plan by modifying existing plan*
    • Reordering steps
    • Removing/replacing steps
    • Changing parameter bindings
    • …

• When Useful?
  – Tweaking or merging existing plans (*case-based planning*)
  – “Planning as Debugging”
    • Can be viewed as “intelligent backtracking”
Partial Plans

- **Complete Plan**
  - Total order, all parameters bound

- **Plan Completion Set**
  - Set of complete plans

- **Partial Plan**
  - Plan consistent with one plan completion set
**Transformational Search Space**

- Search Space of *Refinement* Planners (UCPOP, Prodigy)
  - Can move from one node in search space to another only if completion set of second plan is *subset* of first

- Search Space of *Transformational* Planners
  - No relationship necessary between completion sets of connected nodes in search space
  - Can “jump” between partial plans without backtracking
Advantages and Disadvantages

+ New Search Opportunities
+ Can Avoid Unnecessary Backtracking
+ Can Use Total-Order Planner and Get Many of the Advantages of Partial-Order Planner
+ Can Use for Modifying Existing Plans (plan libraries, case bases)

- Much Larger Branching Factor
- More Complex Algorithms
- Need to Detect Cycles in Search Space
- Hard to Guarantee Completeness
Hacker (Sussman, 1975)

• “A Theory of Skill Acquisition”
  – Either use an existing plan (“subroutine”) from a “library”, or create a new one (either from scratch, or by conjoining and debugging existing plans)

• “The Virtuous Nature of Bugs”
  – Critics look for failures or “un-aesthetic” plans (e.g., moving same object twice in a row)
  – Bugs are patched, then generalized
  – But, patching never leads to wholesale rearrangement

• Cannot Optimally Solve “Sussman’s Anomaly”
  – Linear Planner (no interleaving of learned subroutines)
CHEF (Hammond, 1987)

- Plan Repair for **Case-Based Planning**
  - Build new plans from “memories” (instances) of old ones
  - Tweak plans to fit new situations
- Use **Causal Explanations** to Access Different Repair Strategies
  - Produced by simulation / forward propagation
- Repair Failure Without Interfering with Other Goals
  - Each repair strategy breaks a link in the causal chain
  - Seventeen general repair rules (mostly domain-independent)
    - Reorder events
    - Remove precondition
    - Split step into two and run concurrently
    - Replace existing tool
    - Increase “down” side of a balance relationship
  - Question-answering approach
Making the Perfect Soufflé

- **Strategy**: Trying to make strawberry soufflé by adapting regular soufflé recipe
- **Observe**: Soufflé is *flat*

<table>
<thead>
<tr>
<th>Failure</th>
<th>Side-Effect</th>
<th>Disabled-Condition</th>
<th>Balance</th>
</tr>
</thead>
</table>

**Evaluate Strategy**: Alter-Plan:Precondition

**Question**: Is there an alternative to “bake batter for 25 minutes” that will satisfy “batter now risen” and does not require “thin liquid in bowl from strawberries”?

**Response**: None

**Evaluate Strategy**: Alter-Plan:Side-Effect

**Question**: Is there an alternative to “pulp strawberries” that will enable “dish tastes like berries” and does not cause “thin liquid in bowl from strawberries”?

**Response**: Use “strawberry preserves” instead

**Evaluate Strategy**: Recover

**Question**: Is there a plan to recover from “thin liquid in bowl from strawberries”?

**Response**: After “pulp strawberries” do “drain strawberries”

- Heuristic (domain-specific) knowledge used to choose which repair to actually use for a given failure
Gordius (Simmons, 1987)

• Debug Plans Produced by “Case-Based” Systems
  – Debugging “almost right” plans
  – Also “planning as debugging”: Debug initially null plan

• Analyze Causal Explanations for Bugs
  – Bug is an inconsistency between desired and predicted (observed) state of the world
  – Bug manifestation indicates underlying error in problem solving
  – Can change predicted to match desired, or vice versa

• Assumption-Oriented Repair Strategy
  – Trace causal explanation to assumptions underlying bugs
  – Replace potentially faulty assumptions
  – Regress desired state to determine how to change plan
Debugging Sussman’s Anomaly

**Initial State**

- On(A, B, start)
- On(B, C, start)
- ~On(A, B, start)
- ~On(B, C, start)

**Goal**

- A
- B
- C

<table>
<thead>
<tr>
<th>Bug</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired</td>
<td>On(A, B, end)</td>
</tr>
<tr>
<td>Predicted</td>
<td>~On(A, B, end)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Desired</td>
<td>On(B, C, end)</td>
</tr>
<tr>
<td>Predicted</td>
<td>~On(B, C, end)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Debugging Sussman’s Anomaly

<table>
<thead>
<tr>
<th></th>
<th>Bug</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired</td>
<td>Clear(A, t1)</td>
<td>{Move1}</td>
</tr>
<tr>
<td>Predicted</td>
<td>~Clear(A, t1)</td>
<td>{start &lt; t1, persistence, move1.arg1 = A}</td>
</tr>
<tr>
<td>Desired</td>
<td>On(B, C, end)</td>
<td>{Goal}</td>
</tr>
<tr>
<td>Predicted</td>
<td>~On(B, C, end)</td>
<td>{start &lt; end, persistence}</td>
</tr>
</tbody>
</table>

Planning, Execution & Learning: Transformational

Simmons, Veloso : Fall 2001
Debugging Sussman’s Anomaly

Planning, Execution & Learning: Transformational

Simmons, Veloso : Fall 2001
TPOP (Younes)

- Transformational Partial Order Planner
  - Built on top of UCPOP
  - Record *reasons* for adding constraints to plan (links, bindings, orderings, actions, …)
  - Add transformational operators as threat resolution mechanisms
    - Relink
    - Reorder (*may be exponential # of changes that can undo constraint*)
    - Alter bindings
  - Need to propagate changes if reasons no longer valid
  - Need to avoid cycles in search space
  - Very much dependent on good search heuristics

• **Work in Progress**
Structured Reactive Controllers (Beetz)

- Create Reactive Controllers that are “Transparent”
  - RPL: Expressive, high-level programming language specialized for reasoning about plan execution
    
```lisp
(with-policy (check-signposts-when-necessary)
  (partial-order
    (top-level
      (:tag command-2
        (seq (go 2 2)
          (let ((obs (look-for '((category ball) 0)))
              (if (not (null obs))
                (start-tracking (first obs) 0)
                (fail))))))))
```

- Planner Detects Failures (Real or Simulated) and Debugs
  - Library of plan revisions (XFRML)
  - Uses Monte-Carlo simulation of plans to deal with uncertainty (execution, sensing, environment)
  - Analyzes execution trace to understand bug
**Repairing Reactive Procedures**

- **Plan-Transformation Rules**
  - "**If** a goal might be clobbered by a robot action, **then** execute the clobbering subplan before achieving the goal"
  - "**If** a goal might be left unachieved because the robot overlooked an object, **then** use a different sensing routine for perceiving the object"
  - "**If** a goal might be left unachieved because the robot had an ambiguous object description, **then** achieve the goal for all objects satisfying that description"
  - "**If** a goal might be clobbered by an exogenous event, **then** stabilize the goal immediately after achieving it"
    - **If** GOAL(OB) is clobbered by an exogenous event and DESIG is the data structure returned by the sensing routine that saw OB and the robot tried to achieve GOAL(DESIG) with plan P
    - **Then** replace P with SEQ(P, STABILIZE(GOAL(DESIG)))
Open Questions

• Can we create simple transformational planning algorithms?

• Can we create provably sound and complete transformational planning algorithms

• Under what circumstances do transformational planners perform better than pure refinement planners?

• Can we improve efficiency by combining assumption-oriented approach (GORDIUS, TPOP) with fault-type approach (HACKER, CHEF)?