Challenges in Finding Generalized Plans

Siddharth Srivastava
Joint work with Neil Immerman and Shlomo Zilberstein
University of Massachusetts, Amherst

ICAPS 2009 Workshop on
Generalized Planning: Macros, Loops, Domain Control
September 20th, 2009
Generalized Planning

Plans or planning structures that “work in many situations”

- Triangle Tables [Fikes et al., 1972]
- Case Based Planning [Hammond, 1986]
- Explanation Based Planning [Minton et al., 1989, Shavlik, 1990]
- Contingent Planning
- Learning domain specific planners from examples [Winner and Veloso, 2003]; Planning with loops [Levesque, 2005];
Overview

- Universal Challenges
- Our Framework
- Generalized Planning with Sensing Actions
- Results
Examples of Generalized Plans

Classical Plans

mvToTable($b_3$), mvToTable($b_2$), mvToTable($b_1$)
Examples of Generalized Plans

Classical Plans

\[ \text{mvToTable}(b_3), \text{mvToTable}(b_2), \text{mvToTable}(b_1) \]

More General:

“Unstack”:

\[ \text{while}(\exists b: \text{topmost}(b) \land \neg \text{onTable}(b)) \{ \text{mvToTable}(b) \} \]
Examples of Generalized Plans

Classical Plans
mvToTable(b₃), mvToTable(b₂), mvToTable(b₁)

More General:
“Unstack”:
while(∃b: topmost(b)∧¬ onTable(b)) {mvToTable(b)}

Still More General:
FF, SATPLAN, SGPLAN, ...
Examples of Generalized Plans

Classical Plans
\[ \text{mvToTable}(b_3), \text{mvToTable}(b_2), \text{mvToTable}(b_1) \]

More General:
"Unstack":
while(∃b: topmost(b) ∧ ¬onTable(b)) \{mvToTable(b)\}

Still More General:
FF, SATPLAN, SGPLAN, . . .

Common fundamental problem (Generalized Planning):
Find a function \( G \) (a generalized plan):

\[ G : \text{Problem instance} \rightarrow \text{sequence of actions} \]

What makes us prefer one over another?
Challenges for Any Approach to Generalized Planning

1. Applicability Test
2. Cost of Instantiation
3. Domain Coverage
4. Quality of instantiated plans
5. Complexity of creating generalized plans
### Applicability Test

\[ G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots, a_n \]

- One approach: simulated execution.
- Cost of instantiation will be wasted if \( G \) cannot solve it.

#### NavigateGrids /*Start at bottom left*/

```
repeat
  while ¬rightmost do
    |   mvR()
  end
  mvU()
  mvL()
  while ¬leftmost do
    |   mvL()
  end
  mvU()
until atgoal
```
Applicability Test

\[ G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots a_n \]

- One approach: simulated execution.
- Cost of instantiation will be wasted if \( G \) cannot solve it.

NavigateGrids /*Start at bottom left*/

repeat

| while \( \neg \text{rightmost} \) do
| \hspace{1cm} mvR()
| end
| mvU()
| while \( \neg \text{leftmost} \) do
| \hspace{1cm} mvL()
| end
| mvU()

until atgoal
Historically not common: not required for very general (FF) or very simple plans \((a_1, \ldots a_n)\).

Computed generalized plans typically have a limited applicability.

More of a problem with compact representations (loops).

Simulated execution may not even terminate!!

Ideal applicability test: linear in the size of the problem
Cost of Plan Instantiation

\[ G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots, a_n \]

- Makes generalized plans like “unstack” \( (O(n)) \) more desirable than classical planners \( (O(\exp(n))) \).
- In hindsight: low COI = one of the main motivations behind this field.
Domain Coverage

The set/fraction of solvable problems solved by a generalized plan.

- Historically one of the most measured attributes.
- Trade-offs with cost of instantiation.
Quality of Instantiated Plans

The computational cost (makespan/number of actions/time etc.) of executing the instantiated plan.

- Satisficing, optimal generalized plans.
- Trade-offs with domain coverage and cost of instantiation.
Complexity of Creating Generalized Plans

Universal Challenges  Our Framework  Algorithm  Results

Serious problems with applicability test, instantiation:
Loop termination, progress

Siddharth Srivastava

Challenges in Finding Generalized Plans
Complexity of Creating Generalized Plans

- Serious problems with applicability test, instantiation:
  - Loop termination, progress
Our Objective

- Compute algorithm-like “generalized” plans.
  - Low cost of instantiation
  - Efficient applicability tests
  - Efficient generation of generalized plans
- Need to determine progress and termination.
Concrete States as Logical Structures

\[ \mathcal{V} = \{ \text{object}^1, \text{bin}^1, \text{isGlass}^1, \text{isPaper}^1, \text{in}^2, \text{empty}^1, \text{collected}^1, \text{forGlass}^1, \text{forPaper}^1 \} \]

\[
\begin{align*}
S_1 & \quad S_2 \\
((\text{object}(2))) &= 1 \\
((\text{isPaper}(2))) &= 1 \\
((\text{bin}(1))) &= 1 \\
((\text{in}(2,1))) &= 1 \\
((\text{isGlass}(2))) &= 1 \\
((\text{bin}(1))) &= 1 \\
((\text{in}(2,1))) &= 1
\end{align*}
\]
Example: The Collect Action

\[ \text{Collect}(o, c) \]

\[ \text{object}(o) \land \text{container}(c) \land (\text{isGlass}(o) \leftrightarrow \text{forGlass}(c)) \land \exists b (\text{bin}(b) \land \text{in}(o, b) \land \text{robotAt}(b)) \]

\[ \text{in}'(u, v) := (\text{in}(u, v) \land u \neq o) \lor (\neg \text{in}(u, v) \land u = o \land v = c) \]

\[ \text{empty}'(u) := (\text{empty}(u) \land u \neq c) \lor \text{in}(o, u) \]

\[ \text{collected}'(u) := \text{collected}(u) \lor o = u \]
Abstraction Using 3-Valued Logic

Use 3-Valued logic to abstract as:

TVLA: [Sagiv et al., 2002]
Abstraction Using 3-Valued Logic

Integrity Constraint:
Objects are either paper or glass

Implementation of “sensing” actions
Abstraction Using 3-Valued Logic

Canonical Abstraction

Concretization

Integrity Constraint:
Each bin has a unique object

= "summary" element
TVLA [Sagiv et al., 2002]: Three Valued Logic Analysis

- **Abstraction predicates**: unary predicates.
- Element’s **role** = set of abstraction predicates satisfied
- Collapse elements of a role into **summary elements**.
- Use **integrity constraints** to retrieve concrete states.
- Make structures precise by creating possible cases: focus (automatic)
- Apply action

![Diagram showing the application of action on belief states]

- Role
- $S_0$
- $S_1$
- $S_2$
- $S_3$
- Make structures precise by creating possible cases: focus (automatic)
- Apply action

![Diagram of belief states and actions]

**Action Application on Belief States**

- Challenges in Finding Generalized Plans
Branches solve only *some* members of abstract structures

- May be classifiable, e.g. \( \#_R \{S\} > 1 \)
  - Extended-LL domains: all branches are classifiable
- Subtract role-count changes to obtain preconditions at start.
- Generalize to simple loops, nested loops due to shortcuts and sensing actions.
Plan Generalization

Use abstract structures to recognize loop invariants in example concrete plans.

**Example Execution**

2 objects of each type collected; 2 bins remaining

- $S_0 \xrightarrow{\text{goToNextBin}} S_1 \xrightarrow{\text{senseType}} S_2 \xrightarrow{\text{preProc-Paper}} S_3 \xrightarrow{\text{collectPaper}} S_4$

**Find Loops**

Developed for completely observable settings [Srivastava et al., 2008]
A single plan may not explore all possibilities.

Construct problem instances from unsolved belief states.

Solve them using classical planners.
Example Results

\[ p_0 = \| \{ \text{paper, collected} \} \|; \quad pc_0 = \| \{ \text{empty, container, for paper} \} \|; \]
\[ g_0, gc_0 : \text{similar for glass}; \quad b_0 = \| \{ \text{bin} \} \| \]

Loop 1

- goToNextBin()
- senseType()
- paper
- apply-PaperPreProc(obj)
- collect-Paper-Cont(obj)

- senseType()
- paper
- apply-PaperPreProc(obj)
- collect-Paper-Cont(obj)

- Precons: \( pc_0 = l_1; b_0 = l_1 \)
- Solves 1 out of \( 2^n \)

Loops 1 & 2

- goToNextBin()
- senseType()
- apply-GlassPreProc(obj)
- collect-Glass-Cont(obj)
- goToNextBin()
- senseType()
- paper
- apply-PaperPreProc(obj)
- collect-Paper-Cont(obj)

- Precons:
  \[ pc_0 = l_1; gc_0 = l_2; b_0 = l_1 + l_2 \]
  \[ 2^{n-1} + 1 \text{ out of every } 2^n \]
Merging Generalized Plans: Algorithm

Input: Existing plan \( \Pi \), eg trace \( \text{trace}_i \)

Output: Extension of \( \Pi \)

1 if \( \Pi = \emptyset \) then
2 \( \Pi \leftarrow \text{trace}_i \)
3 return \( \Pi \)
end

4 \( mp_{\Pi}, mp_t \leftarrow \text{findMergePoint}(\Pi, \text{trace}_i, bp_{\Pi}, bp_t) \)
5 repeat
6 if \( mp_{\Pi} \) found then
7 \( bp_{\Pi}, bp_t \leftarrow \text{findBranchPoint}(\Pi, \text{trace}_i, mp_{\Pi}, mp_t) \)
end
8 if \( bp_{\Pi} \) found then
9 \( mp_{\Pi}, mp_t \leftarrow \text{findMergePoint}(\Pi, \text{trace}_i, bp_{\Pi}, bp_t) \)
10 \( \text{addEdges}(\Pi, \text{trace}_i, bp_t, mp_t, mp_{\Pi}, bp_{\Pi}) \)
end
until new \( bp_{\Pi} \) or \( mp_{\Pi} \) not found

11 return \( \Pi \)

Algorithm 1: ARANDA-Merge
Addressing the Challenges

- Cost of testing applicability: independent of the size of the problem.
- Cost of instantiation: linear, or better with role-lists
- Domain Coverage can increase exponentially with new examples
- Complexity of creating generalized plan: $O(s \cdot n_{eg}^2)$ to find loops, $O(s \cdot n_{eg})$ for preconditions.
Clear formal framework for algorithmic plans, avoiding intractability of automated program synthesis.

Approach for learning generalized conditional plans with nested loops by composition of simple linear plans.

Efficient methods for computation of measures of progress and preconditions.
Transport Domain
Transport Domain: Results

\[ m_0 = ||\{\text{monitor, atD2}\}||; s_0 = ||\{\text{server, atD1}\}|| \]

**Loop 1**

- Precons: \( m_0 = l_1; s_0 = l_1 \)

**Loops 1 & 2**

- Precons: \( m_0 = l_1; s_0 = l_1 + k_1 \)
Example Results: Domain Coverage

\[ D_\pi(n) = \frac{|S_\pi(n)|}{|T(n)|} \]

Siddharth Srivastava  Challenges in Finding Generalized Plans
Related Work

- Plans with Loops
  - [Winner and Veloso, 2007]: no preconditions or sensing actions, but use partial ordering.
  - [Levesque, 2005]: single planning parameter, limited preconditions.
  - [Cimatti et al., 2003]: “hard” loops.

- Planning with unknown quantities:
  - [Milch et al., 2005]: action operators not provided.
References I


References II


References III

