Planning with Unknown Object Quantities and Properties

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

Eighth Symposium on Abstraction, Reformulation and Approximation
7 – 10 July, 2009
Overview

- Introduction

- Framework
  - Concrete Representation
  - Abstract Representation for Belief States
  - Actions on Belief States

- Planning Algorithms
  - Plan Generalization
  - Plan Merging
  - Preconditions

- Results
Conditional Planning

Test Type

<table>
<thead>
<tr>
<th>Test Type</th>
<th>collect In PaperContainer</th>
<th>collect In GlassContainer</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper</td>
<td>pick Next</td>
<td>pick Next</td>
</tr>
<tr>
<td>glass</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Planning with Unknown Object Quantities and Properties

Siddharth Srivastava
Conditional Planning

Need to merge, maintain history
Need to merge, maintain history

Progress and termination of loops
Concrete States as First-Order Structures

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

\[ \begin{align*}
S_1 & \quad \text{with } \begin{cases}
((object(2)) = 1 \\
((isPaper(2)) = 1 \\
((bin(1))) = 1 \\
((in(2,1))) = 1
\end{cases} \\
S_2 & \quad \text{with } \begin{cases}
((object(2)) = 1 \\
((isGlass(2)) = 1 \\
((bin(1))) = 1 \\
((in(2,1))) = 1
\end{cases}
\end{align*} \]
Action Operators

- Precondition: formula in FO.
- Predicate updates

\[ p'(\bar{x}) = \begin{cases} 
\text{tuples added to } p & \exists \Delta^+_p(\bar{x}) \\
\text{tuples retained in } p & \Delta^-_p(\bar{x}) 
\end{cases} \]

- Use formula evaluation to compute action effect.
- Frame axioms/successor state axioms (situation calculus) using a double vocabulary.
Example: The Collect Action

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

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

\[
\begin{align*}
\text{in}'(u, v) & := (\text{in}(u, v) \land u \neq o) \lor \\
& \quad (\neg \text{in}(u, v) \land u = o \land v = c) \\
\text{empty}'(u) & := (\text{empty}(u) \land u \neq c) \lor \text{in}(o, u) \\
\text{collected}'(u) & := \text{collected}(u) \lor o = u
\end{align*}
\]
Abstraction Using 3-Valued Logic

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

Siddharth Srivastava
Planning with Unknown Object Quantities and Properties
Abstraction Using 3-Valued Logic

Canonical Abstraction

= Summary element
Abstraction Using 3-Valued Logic

Concretization

Integrity Constraint:
Each bin has a unique object
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.
Action Application on Belief States

Siddharth Srivastava
Planning with Unknown Object Quantities and Properties
A simpler case: only paper objects and containers

Underspecified action application
Drawing Out Action Arguments

\( \phi \) constrained to be unique and satisfiable
\( \phi \) constrained to be unique and satisfiable
Summary of Action Application

- Draw-out action arguments prior to application
- Use focus and coerce to create cases for properties of drawn out of objects.
- Branches caused:
  - Classifiable, e.g. \( #_R \{S\} > 1 \)
  - Unpredictable at planning-time, e.g. “object type=paper”
Plan Generalization

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

Example Execution

2 objects of each type collected; 2 bins remaining

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

Tracing

\[ S_0 \xrightarrow{\text{goToNextBin}} S_1 \xrightarrow{\text{preProc-Paper}} S_3 \xrightarrow{\text{collectPaper}} S_0 \]

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.
Branches solve only some members of abstract structures.
Classify branches using role counts; propagate backwards.
Need for doing this constrains predicate update formulas.
Example Results

Initial: \( p_0 = \|\{\text{paper, collected}\}\|; p_{c0} = \|\{\text{empty, container, for paper}\}\|; \)
\( g_0, g_{c0} : \text{similar for glass}; b_0 = \|\{\text{bin}\}\| \)

Loop 1

- Final: \( p_0 + l_1; p_{c0} - l_1; b_0 - l_1 \)
- Solves 1 instance out of every \( 2^n \)

Loops 1 & 2

- Final: \( p_0 + l_1; p_{c0} - l_1; g_0 + l_2; g_{c0} - l_2; b_0 - l_1 - l_2 \)
- \( 2^{n-1} + 1 \) out of every \( 2^n \)
Transport Domain

D1 L D3

D2

T1: Capacity 1
T2: Capacity 2
Transport Domain: Results

Initial undelivered counts:

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

**Loop 1**

- load(server, T1) → mv(T1, L) → unload(T1)
- mv(T1, D1)
- load(LCD, T2)
- mv(T2, L)
  - server present
- load(server, T2)
- mv(T2, D3) → unload(T2) → mv(T2, L)
- mv(T2, D2)

**Final:** \( m_0 - l_1; s_0 - l_1 \)

**Loops 1 & 2**

- load(server, T1) → mv(T1, L) → unload(T1)
- mv(T1, D1)
- load(LCD, T2)
  - server lost
- mv(T2, D1) → load(server, T2)
- mv(T2, L)
- load(server, T2)
- mv(T2, D3) → unload(T2) → mv(T2, L)
- mv(T2, D2)

**Final:** \( m_0 - l_1; s_0 - l_1 - k_1 \)
Solutions: Recycling and Transport

Recycling:
- goToNextBin()
- senseType()
- apply−PaperPreProc(obj)
- apply−GlassPreProc(obj)
- collect−Glass−Cont(obj)
- collect−Paper−Cont(obj)
- goToNextBin()
- apply−PaperPreProc(obj)
- apply−GlassPreProc(obj)
- senseType()
- apply−PaperPreProc(obj)
- collect−Glass−Cont(obj)
- collect−Paper−Cont(obj)

Transport:
- mv(T2, L) → mv(T2, D2)
- load(kind1, T1)
- mv(T1, L)
- unload(T1) → mv(T1, D1) → load(kind2, T2)
- mv(T2, L)
- unload(T2)
- load(kind1, T2)
- forkLift(kind1, T2)
- mv(T2, D1)
- forkLift(kind1, T1)
- load(kind1, T1)
- mv(T2, D3)
- forkLift(kind1, T2)
- mv(T2, L)

Siddharth Srivastava
Planning with Unknown Object Quantities and Properties
Conclusions

- An approach for representing unknown quantities for planning.
- Methods for finding generalized plans with branches and loops.
- Automatic computation of preconditions for many kinds of nested loops in a broad class of domains.
Example Results: Domain Coverage

\[ D_{\pi}(n) = \left| S_{\pi}(n) \right| / \left| T(n) \right| \]
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. repeat

5. $\text{mp}_\Pi, \text{mp}_t \leftarrow \text{findMergePoint}(\Pi, \text{trace}_i, \text{bp}_\Pi, \text{bp}_t)$

6. if $\text{mp}_\Pi$ found and not first iteration then

7. $\text{attachEdges}(\Pi, \text{trace}_i, \text{bp}_t, \text{mp}_t, \text{mp}_\Pi, \text{bp}_\Pi)$

end

8. if $\text{mp}_\Pi$ found then

9. $\text{bp}_\Pi, \text{bp}_t \leftarrow \text{findBranchPoint}(\Pi, \text{trace}_i, \text{mp}_\Pi, \text{mp}_t)$

end

until new $\text{bp}_\Pi$ or $\text{mp}_\Pi$ not found

10. return $\Pi$

**Algorithm 1:** ARANDA-Merge
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.
Plan Generalization: Example

```
findDest(); Load(); setTarget(destLoc); Drive(); Unload(); Choose(Crate)
```
Plan Generalization: Example

Consider the following scenario:

1. **S1**: Dock at destination, at Truck, at Garage.
   - Delivered; crate

2. **S2**: Dock at destination, at Truck, at Garage.
   - Delivered; crate

3. **S3**: Dock at destination, at Truck, at Garage.
   - Delivered; crate

4. **S4**: Dock at destination, at Truck, at Garage.
   - Chosen; crate

Choose (crate) to select the appropriate action.


