Finding Plans with Branches, Loops, and Preconditions

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

ICAPS 2009 Workshop on
Verification and Validation of Planning and Scheduling Systems
September 20th, 2009
Overview

- Introduction
- Our Framework
- Planning Algorithms
- Results
Conditional Planning

Finding Plans with Branches, Loops, and Preconditions
Serious problems with applicability test, instantiation:
  - Loop termination, progress
One approach: simulated execution.
Will be wasted if $G$ cannot solve a problem.

NavigateGrids /*Start at bottom left*/

```
repeat
   while ¬rightmost do
      mvR()
   end
   mvU()
   while ¬leftmost do
      mvL()
   end
   mvU()
until atgoal
```
Plan Preconditions

- One approach: simulated execution.
- Will be wasted if \( G \) cannot solve a problem.

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

```plaintext
repeat
    while ¬rightmost do
    |    mvR()
    end
    mvU()
    mvU()
    while ¬leftmost do
    |    mvL()
    end
    mvU()
until atgoal
```
Historically not common: not required for simple plans \((a_1, \ldots a_n)\).

Computed plans with loops etc. will typically have a limited applicability.

- Simulated execution may not even terminate!!

Ideal applicability test: linear in the size of the problem
Our Objective

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

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

**Collect(o,c)**

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

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

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

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

Use 3-Valued logic to abstract as:

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

Focus and coerce w.r.t \{isPaper(x)\}

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.
Action Application on Belief States

- Make structures precise by creating possible cases: focus (automatic)
- Apply action
Action Application on Belief States

- Make structures precise by creating possible cases: focus (automatic)
- Apply action
Role-counts, Branches and Plan Preconditions

Changes in role-counts:

- \( \{\text{obj, atL1}\} \) increases
- \( \{\text{obj, atL2}\} \) decreases
- \( \{\text{obj, atL1}\} \) decreases
- \( \{\text{obj, atL2}\} \) increases
- \( \{\text{obj, atL1}\} \) remains unchanged

- \( \{\text{obj, atL2}\} \) remains unchanged

Goal is provably reachable from the infinitely many structures represented by \( S_1 \).

\( \forall s \in S_1 \), can compute number of steps required to reach the goal.

Generalized to extended-LL domains.

Siddharth Srivastava
Finding Plans with Branches, Loops, and Preconditions
Goal is provably reachable from the infinitely many structures represented by $S_1$.

\( \forall s \in S_1 \), can compute number of steps required to reach the goal.

Generalized to extended-LL domains.
Extension to Complex Loops with Shortcuts

Shortcuts due to sensing actions

(a) A simple loop
(b) A simple loop with (non-composable) shortcuts
Extension to Nested Loops?

for i = 1 to n {
    for j = 1 to k {
        if (...) {
            if (...) {
                ...
            }
        }
    }
}

for i = 1 to n {
    if (...) {
        if (...) {
            ...
        }
    }
}
Extension to Nested Loops?

for i = 1 to n {
    for j = 1 to k {
        if (...) {
            if (...) {
                
            }
        }
    }
}

for i = 1 to n {
    if (...) {
        if (...) {
            
        }
    }
}

Methods applicable to many so-called "nested" loops.

Translation of the loop entry point $\Rightarrow$ complex loop with a shortcut!

Execution sequences match.
### Extension to Nested Loops?

- **Execution sequences match.**
- **Translation of the loop entry point** $\Rightarrow$ complex loop with a shortcut!
- **Methods applicable to many so-called “nested” loops.**
Plan Generalization

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

**Example Execution**

2 objects of each type collected; 2 bins remaining

```
S0 # goToNextBin() S1 # senseType() S2 # preProc-Paper() S3 # collectPaper() S4
```

**Find Loops**

Developed for completely observable settings [Srivastava et al., 2008]
Merging Generalized Plans

Plan for Unhandled Structure

A single plan may not explore all possibilities.

Construct problem instances from unsolved belief states.

Solve them using classical planners.
Example Results

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

**Loop 1**

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

**Loops 1 & 2**

- **Precons:**
  \[ pc_0 = l_1; gc_0 = l_2; b_0 = l_1 + l_2 \]
- **Solves \( 2^{n-1} + 1 \) 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. $bp_{\Pi}, bp_t \leftarrow \text{findBranchPoint}(\Pi, \text{trace}_i, mp_{\Pi}, mp_t)$
8. end
9. if $bp_{\Pi}$ found then
10. $mp_{\Pi}, mp_t \leftarrow \text{findMergePoint}(\Pi, \text{trace}_i, bp_{\Pi}, bp_t)$
11. $\text{addEdges}(\Pi, \text{trace}_i, bp_t, mp_t, mp_{\Pi}, bp_{\Pi})$
12. end
13. until new $bp_{\Pi}$ or $mp_{\Pi}$ not found

14. return $\Pi$

**Algorithm 1:** ARANDA-Merge
Conclusions

- Approach addressing plan/algorith synthesis and verification.
  - Advantage of automated synthesis: can choose to keep control structure verifiable.
- Close to program synthesis, but free of associated intractability.
- Efficient precondition tests and measures of progress.
Transport Domain

D1

L

D3

D2

T1: Capacity 1

T2: Capacity 2
$m_0 = \|\{\text{monitor, atD2}\}\|; s_0 = \|\{\text{server, atD1}\}\|$
Example Results: Domain Coverage

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


