Learning Generalized Plans Using Abstract Counting

Siddharth Srivastava, Neil Immerman, Shlomo Zilberstein

Twenty Third Conference on Artificial Intelligence
17th July, 2008
Overview

- Introduction

- Our Approach
  - Abstraction Mechanism
  - Algorithm for Learning Generalized Plans

- Results

- Conclusions
**Plans vs Algorithms**

[Diagram showing plans and algorithms involving storage, transportation, and manufacturing processes.]
Plans vs Algorithms

Move Truck to Dock
While #(undelivered crate)>0
    Load a crate
    Find crate’s destination
    Move truck to destination
    Unload crate
    Move Truck to Dock
Move Truck to Garage

Load a crate
Find crate’s destination
Move truck to destination
Unload crate
Move Truck to Dock
Move Truck to Garage
Finding Algorithm-like Plans

Variants of this problem have been of continued interest.

Recurring Hurdles

- Problem definition: unknown numbers
- Plans with loops: finding loops
- Plans with loops: reasoning about loops (Plan correctness)

Myth  Systematic approach  undecidable  (cf. automated programming)

We identify a tractable piece of this problem.
Generalized Plans

A formalization of algorithm-like plans.

- Connected, directed graph.
- Nodes → actions.
- Edges → conditions.
- Start/terminal nodes.
Our Approach

- Learn from an example plan
- Recognize loops through loop invariants
- Use abstraction to identify similar states for determining invariants
Representation: States as Logical Structures

Dock(1)
Garage(2)
Truck(3)
Crate(6)
Crate(10)
at(3,1)
\vdots
delivered(10)

Integrity constraints specify legal structures.

\[ \nu = \{ \text{Garage}^1, \text{Dock}^1, \text{Loc}^1, \text{Truck}^1, \text{Crate}^1, \text{delivered}^1, \text{at}^2, \text{in}^2, \text{dest}^2 \} \]

\[ |S| = \{1, 2, \ldots, 10\} \]
Representation: Actions

- **Precondition**: formula in FO(TC).
- **Action operators** = structure transformers
  - Predicate updates
    - $p'(\bar{x}) = (\neg p(\bar{x}) \land \Delta_p^+(\bar{x})) \lor (p(\bar{x}) \land \neg \Delta_p^-(\bar{x}))$

**mv(A,B):**

$$topmost'(x) = (\neg topmost(x) \land on(A,x)) \lor (topmost(x) \land x \neq B).$$
Review: Need for Abstraction

Idea: collapse similar states together.

- Makes identifying invariants (recurring properties) easy.
- Use an abstraction mechanism.

We use an abstraction scheme from static analysis.
Abstraction Using 3-Valued Logic

TVLA [Sagiv et al., 2002]: Three Valued Logic Analysis

- **Abstraction predicates**: chosen unary predicates.
- Values of all abstraction predicates on an element define its role.
- Collapse elements of the same role into summary elements.
- Relations involving summary elements may become indefinite.

States from infinitely many instances $\mapsto$ finite set of abstract states
Precision in Action Updates

Predicate update formula:

\[ p'(\bar{x}) = (\neg p(\bar{x}) \land \Delta^+_p) \lor (p(\bar{x}) \land \neg \Delta^-_p) \]

TVLA’s focus+coerce operations: make structure precise wrt a user defined formula (automatically determined in our approach).

\( \phi \) constrained to be unique.

Use this for sensing actions too.
Learning Generalized Plans

We recognize loop invariants by tracing example plans in the abstract state space.

**Algorithm for Learning Generalized Plans**

- Change action arguments to their roles in the example plan.
- Apply resulting plan to abstraction of the given start state.
- Find loops in the resulting state and action sequence.
Tracing

findDest(); Load(); setTarget(destLoc); Drive(); Unload(); Choose(Crate)
Tracing

\[\text{findDest}(); \text{Load}(); \text{setTarget}(); \text{Drive}(); \text{Unload}(); \text{Choose}() = \text{delivered, Crate}\]
Tracing
Finding Preconditions

In generalized planning, correctness $\equiv$ applicability.

- **Classify** branches on the basis of *role counts*; **propagate** these counts backwards.

- Need for doing this constrains predicate update formulas.

\[
\begin{align*}
R & \xrightarrow{f_\phi} R \\
S_0 & \quad \phi \quad \phi \\
S_\phi & \quad \phi \quad \phi \\
S_1 & \quad \phi \\
S_2 & \quad \phi \\
\#R &= 1 & \#R &> 1
\end{align*}
\]

$\phi$ constrained to be unique and satisfiable.
Finding Preconditions

In generalized planning, correctness $\equiv$ applicability.

- **Classify** branches on the basis of *role counts*; **propagate** these counts backwards.

- Need for doing this constrains predicate update formulas.

$\phi$ constrained to be unique and satisfiable
Problem Domains

Delivery

Assembly and Transport

Striped Block Tower

(a) (b) (c)
Results: Delivery

Learned plan for unit delivery
Results: Transport

```
#(monitor atL1) = 1
mvToL3()
mvToL2()
chooseVehicle(Van)
chooseItem(monitor;atL1)
Load()
mvToL3()
Unload()
mvToL1()
chooseVehicle(Truck)
mvToL2()
chooseItem(server;atL2)
Load()
mvToL3()
chooseItem(monitor;atL3)
Load()
mvToL4()
chooseItem(server;atL2)
Load()
mvToL3()
chooseItem(monitor;atL3)
Load()
mvToL4()
```

Learned plan for transport
Results: Blocks
Results: Running Times

Execution Time Breakups

- Delivery
- Transport
- Blocks

Problem

0
5
10
15
20
25
30
35
40

Time (s)

Planning Times

Transport: SGPlan5
SATPLAN06
Aranda
Blocks: SGPlan5
SATPLAN06
Aranda

Number of Items of Each Kind

0  5  10  15  20  25  30  35  40

Time (s)
Conclusions

- Novel algorithm for generalizing plans and finding loops.
- Identified a class of domains where our methods are proven to work (extended-LL).
- No need for plan annotations/parameterization etc.

Work in Progress/Future Directions

- Plan synthesis
- Extensions beyond extended-LL domains
- Plan evaluation.
Existing Approaches

Other research along this direction

- Plan compilation: Triangle tables [Fikes et al., 1972], case based planning [Hammond, 1989]
- Explanation based learning of plans (BAGGER2) [Shavlik, 1990]
- Extracting plan templates (DISTILL) [Winner et al., 2003], planning with loops (KPLANNER) [Levesque, 2005]
Extended-LL Domains

Look like linked lists upon abstraction.

**Theorem**

*In “extended-LL” domains, we can compute all the branch conditions and propagate them backwards to get preconditions for plans with simple loops.*

We can find complete generalized plans through search in these domains!

- Defined as a set of syntactic constraints on action update formulae making sure that action updates don’t require more precision than is available in abstract structures.
- Predicate change formulas which need focusing are role-specific, uniquely satisfiable.