Iterative Planning: A Survey

Yuxiao (Toby) Hu

yuxiao@cs.toronto.edu

Department of Computer Science
University of Toronto

Toronto, Ontario, Canada

November 18, 2008

Computer Science
UNIVERSITY OF TORONTO

Outline

Outline

Background on Automated Planning

- Automated Planning
  - Given a formal specification of a dynamical system, the initial state and a goal condition, find an action strategy that realizes the goal.
- Different types of planning
  - Classical planning: STRIPS [Fikes *et al.* 71], ADL [Pednault 89];
  - Sequential extensions: numerics and time [Fox and Long 03], temporally extended goals [Gerevini and Long 05], *etc*;
  - Conformant planning [Smith and Weld 98];
  - Conditional planning [Petrick and Bacchus 98; Bertoli *et al.* 98].

Motivation for Iterative Planning

- Planning tasks above are concerned with *individual* problems, *e.g.*
  - Three blocks are on the table. Stack them into a tower!
  - A tree can be felled with no more than two chops. Chop it down!

  What if we have 5 blocks on table or a tree needing 4 chops?
- It is desirable to find a solution to a *class* of problems. Then solving a problem in the class is simply an instantiation of the solution. This requires loops [Levesque 05]!
- Learn from small problems so as to more efficiently solve larger ones.

Plan Representation

Robot Programs

- Ideally, a plan is a deterministic procedure to follow without further deliberation.
- Robot programs are defined inductively for representing loopy plans [Levesque 96]
  1. **nil** is a robot program;
  2. if A is a primitive action and P is a robot program, then $\mathbf{seq}(A, P)$ is also a robot program;
  3. if A is a sensing action with sensing results $R_1, \cdots, R_n$, and $P_1, \cdots, P_n$ are robot programs, then $\mathbf{case}(A, [\mathbf{if}(R_1, P_1), \cdots, \mathbf{if}(R_n, P_n)])$ is also a robot program;
  4. if P and Q are robot programs, and P' is the result of replacing some of the occurrances of **nil** by **exit** and the rest by **next**, then $\mathbf{loop}(P', Q)$ is a robot program.

An Example

```
loop(
    case(look,
            [if(down,exit),
             if(up,seq(chop,next))
            ]
        ),
    seq(store,nil)
)
```

Related Problems

- Program synthesis [Manna and Waldinger 92; 80]

  Given a constraint on valid input $P(x)$ and the relationship between input and output $R(x, y)$, find a program $f(x)$ such that for any input a satifsying $P(a)$, the output $z = f(a)$ satisfies $R(a, z)$.

- Grammar induction [Section 8.7 of Duda *et al.* 01]

  Find the underlying grammar that can generate the observed strings from a language.

- Repeated-attempt problems
  - Pick up block with success probability p [Haddawy and Ngo 95]
  - The probability of getting a good egg is p [Bonet and Geffner 01]

Outline

Deductive Approaches

Generate a loopy plan as a by-product of proving a mathematical theorem.

- Tableau-based sequent calculus [Manna & Waldinger 80; 87]
  - Use an <assertion,goal,output> triple (sequent) to represent theorems
  - A set of derivation rules to obtain correct new sequents
  - Recursion introduced by a well-founded induction rule

Deductive Approaches

- Deduction-based refinement planning [Stephan & Biundo 96]
  - Problem specification and executable plan represented in a unified language
  - Refinement rules to gradually substitute specification with executable plan
  - Loop structure exists in the original non-executable problem specification

Non-Deductive Approaches

KPLANNER: Generate and Test [Levesque 05]

- Solves a class of planning problems parameterized by an integer (plan parameter)
- Generate a loopy plan that works for a small integer N_1 (the generation bound)
  - Exhaustively search for a conditional plan that works for N_1
  - Wind the conditional plan into a loopy plan
- Test the resulting plan with a larger integer N_2 (the test bound)
  - If it passes the test, the plan is returned.
  - If it fails, go back to the generation phase.
- Correctness
  - The returned plan is guaranteed to work for N_1 and N_2 only, but in practice, it usually works for all integers.
  - For problems with certain properties, the returned plan is guaranteed to work in general.

Non-Deductive Approaches

loopDistill: Identifying Regularity in Partial-Order Plans [Winner & Veloso 07]

- Given a partial order plan for a planning problem, find instances of a same action.
- Greedily identify a largest matching subplan by considering neighboring actions.
- Conditionals and loops are constructed from the matching subplans

Non-Deductive Approaches

Role-Based Abstraction [Srivastava *et al.* 08]

- Characterize the role of an object by the truth values of all unary predicates applied to the object.
- Given a concrete plan for an example problem, construct an abstract plan where objects are replaced by their roles.
- Based on the repetition pattern of actions in the abstract plan, identify loops.
- Guaranteed correctness for extended-LL domains.

Non-Deductive Approaches

Explanation-Based Generalization

- BAGGER2 generates recursive concepts as explanation-based learning with the ability of "generalization-to-N" [Shavlik 90].
- With a similar idea, [Schmid & Wysotzki 00] learns recursive macro operators for planning domains.
  - Predefined data-type structures (natural numbers, lists, sets, *etc.*);
  - Explore problems of small complexity to generate loops that work for all, like in KPLANNER.

Summary of Different Approaches

- Deductive approach
  - Provable correctness
  - Slow and may require human expertise
- Non-deductive approach
  - Efficient and automatic
  - Weak guarantee of correctness

Outline

Underlying Theory: Finite Verification

- Finitely verifiable theories have the property that whether a sentence is a theorem can be checked wrt a finite set of models of the theory [Lin 2007].
- Applied to planning domains, to see if a loopy plan works for a finitely verifiable problem
  - Identify the models that is sufficient for the judgment
  - Correctness verified by finite model checking

Underlying Theory: Identification in the Limit

- Provable correctness relies on the assumption that there is a complete characterization of legal initial states.
- When no such complete characterization is available, finding loopy plans resembles "identification in the limit" [Caldon & Martin 07; Gold 67].
  - There is an infinite supply of instances of a concept
  - The goal is to learn the concept
  - The learner has a hypothesis that explains the observations so far
  - The learner revises its hypothesis when it does not explain the newly observed instance
  - The concept is considered learnable if the learner identifies it after finite mind changes

Outline

Possibilities for Future Work

- Find algorithms that
  - are more efficient
  - solve more problems
- Identify classes of problems with provable correctness guarantees
- Applications to learning for planning (IPC learning track)