Learning Generalized Policies in Planning using Concept Languages

Mario Martín
Universitat Politècnica de Catalunya
Barcelona, Spain

Héctor Geffner
Universidad Simón Bolívar
Caracas, Venezuela

Planning: Scaling Up

- In the last few years, substantial progress in planning algorithms (Graphplan, SatPlan, . . .)
- Still **domain-independent planning** is hard, and solving large problems remains a challenge
- Alternative is addition of **control knowledge**; e.g., in blocks-world

  do not move a block if it is well-placed

- Resulting planners tend to be more effective (TLplan,Shop,TalPlan) but coming up with useful control knowledge is **not always easy**

Goals

- In this work, we are concerned with **learning control knowledge** in planning through examples
- Control knowledge can mean many things; a clause in a SAT formulation, a heuristic function, a decomposition rule, a 'hint', . . .
- The type of control knowledge we are interested is such that **makes search unnecessary**
- We want to learn how to solve problems in one domain with no search, given solutions to a small number of problems

Feasibility

- Useful and compact action strategies for solving problems with no search can be given for most planning domains; e.g., in blocks
  - put all *misplaced* blocks on the table
  - stack *misplaced* blocks on target block when target blocks *well placed*
- Strategy is **not optimal**, but resulting lengths not bad
- Similar suboptimal but useful action strategies can be constructed for other domains
- The question we want to address is **how to learn such action strategies automatically**

What to learn

- The **mathematical object** we are after is a **function** π_D that given a **domain instance** selects an **action**
- A **domain** D is assumed to be given by a set of action schemas and predicates
- An domain **instance** I is given by
  - a domain
  - a set of (typed) objects
  - an initial state
  - a goal
- The function π_D that we seek has the form:

$$\pi_D \;:\; I \longrightarrow A$$

- The **plan** to solve an instance I is given then by
  - the action $a_0 = \pi_D(I)$, followed by
  - the action $a_1 = \pi_D(I')$, where $I' = I$ except for s_0
  - ...and so on, until goal is reached

Generalized Policies

- The functions π_D that map instances into actions are called **policies**
- It is known that (optimal) solutions of **deterministic** planning problems, and **fully observable stochastic** planning problems, can be expressed by such policies
- **Dynamic programming** methods can be used for finding such policies but over a **restricted set of instances**
  - instances sharing the same state space (set of objects),
  - normally the same goal, and in which
  - the state space is not too large
- The functions π_D have broader scope: they apply to **any** domain instance. We call them **generalized policies**.

Representation of Generalized Policies

- Problem of learning generalized policies from examples addressed recently by Roni Khardon (AIJ 1999)
- Khardon represents policies by list of quantified rules like

$$clear(y), holding(x), inplace(y), G(on(x,y)) \Rightarrow Stack(x,y)$$

- Conditions evaluated on domain instance (domain, objects, state, goal), and **first** rule/binding that triggers, determines action
- Vocabulary of rules includes
  - the primitive domain predicates (e.g., $holding$, on)
  - **background** predicates (e.g., $inplace$, $above$)
  - the goal marker G (for atoms to be evaluated in goal)

Learning Rule-based Policies

- Khardon learns rules in supervised manner, given a **training set** of domain instance, action pairs
- Rules restricted to max number of conditions and free variables
- The **algorithm** is a version of Rivest's algorithm for learning decision lists that accepts the set of all rules and examples, and produces an **ordered list of rules**

```
1. initialize Output to empty list
2. evaluate all Rules over all Examples
3. determine # of examples covered right/wrong
4. pick best rule and add to Output
5. remove Examples covered by selected rule
6. go back to 3 until no Examples left
```

Results obtained by Khardon

+ Policies obtained from 4800 examples with 8 blocks solve
  - 79% of test problems of same size
  - 60% of test problems with 15 blocks
  - 40% of test problems with 20 blocks
+ Results show **bootstrap** effect: system solves problems that teacher does not
+ Similar results reported for logistic domain

Limitations

- Approach relies on **background knowledge** in the form of **useful domain predicates** (e.g., in_place, $above$, $\ldots$)
- Coverage not as good as for 'naive' block-world strategy which is 100%

In this work we aim to address these limitations

- Learn useful predicates and policies **simultaneously** in **alternative** representation language
- Get better results with **no background knowledge**
- Learn **control knowledge** in **domain-independent** manner

Representing Policies with Concepts

- **Idea:** languages that provide direct support for **classes** may be suitable for representing policies as rules like

  if there is an object in certain class, then do action a on it

  appear common in planning
- **Concept languages** developed in AI capture fragments of FOL in **syntax** suited for reasoning with classes
- Syntax allows **compact** definitions of **complex classes** in terms of primitive ones
- Motivation for concept languages in AI is that **syntax** matters in **reasoning**; the point here is that syntax matters in **learning** too

A Simple Concept Language

- Unary predicates called **concepts** and binary predicates called **roles**
- **Primitive** concepts C_p and roles R_p obtained from domain description
- **Complex** concepts follow from standard grammar

  $$C, C' \longrightarrow C_p \mid \top \mid \bot \mid \neg C \mid C \wedge C' \mid (\forall R.C) \mid R = R'$$

  and **complex** roles follow from

  $$R, R' \longrightarrow R_p \mid R_p^{-1} \mid R_p^* \mid R \circ R'$$
- **Denotation** of complex concepts and roles determined by denotation of primitive ones

  e.g., concept $(on_s = on_g)$ denotes the objects that are on the same location in current state and goal

  $(\forall on_g^*.\, on_s = on_g)$ denotes blocks whose targets are well-placed

Concept-Based Policies

- Simple **concept-based policies** are defined as lists of pairs

$$C_1 : a_1\ ,\ C_2 : a_2\ , C_3 : a_3, \ldots C_n$$

  where C_i are concepts and a_i unary action predicates
- These policies trigger action a_i when C_i is first concept with non-empty denotation C_i^I, and apply a_i to object in C_i^I
- Domains involving **non-unary actions** can be reformulated in terms of unary actions, although this is not always most effective approach
- Not all arguments appearing in Strips descriptions needed

Learning Concept-Based Policies: Experiments

- Rivest's algorithm used for learning policies in blocks-world
- Predicates: $clear_s$, $clear_g$, $holding_s$, on_s, and on_g
- Actions $pick(block)$, $put\_on(block)$, $put\_on\_table(block)$
- **No** additional background predicates used
- Training set obtained by solving 50 random problems
- Complex concepts restricted to a max size (6)
- Number of concept-action pairs is 216.684

Results

- Example of learned policy

$$\begin{array}{l}
\mathrm{PUT_ON}\,((on_g = on_s) \wedge (\forall\ on_g^{-1}.holding) \wedge clear_s) \\
\mathrm{PUT_ON_TABLE}\,(holding) \\
\mathrm{PICK}\,((\forall\ on_g^{*}.(on_g = on_s)) \wedge (\forall\ on_g.clear_s) \wedge clear_s) \\
\mathrm{PICK}\,(\neg(on_g^{*} = on_s^{*}) \wedge (\forall on_s.(\forall on_g^{-1}.clear_s)) \wedge clear_s) \\
\mathrm{PICK}\,(\neg(on_g = on_s) \wedge (\forall\ on_g^{*}.(on_s^{*} = on_s)) \wedge clear_s) \\
\mathrm{PICK}\,(\neg(on_g = on_s) \wedge (\forall\ on_s.(\forall\ on_s^{-1}.clear_s)))
\end{array}$$

- Coverage on test examples

5 blocks	10 blocks	15 blocks	20 blocks
96.5%	87.6%	82.8%	72.2%

- Quality of plans generated

Policy	5 blks	10 blks	15 blks	20 blks
Naive	11.57	27.74	45.19	62.68
Learned	9.98	24.32	39.77	54.94
Optimal	9.94	23.48	37.5	51.60

Further issues

- Policies can be improved by extending training set with test examples for which policy **fails**
- In this way, we have obtained more recently policies with 100% coverage that produce plans that are 10% away from optimal (in problems with up to 50 blocks)

Related ideas and methods

- Learning **rule-based** policies (Khardon)

  requires handcrafted domain features

- Learning policies with **reinforcement learning** (Sutton, Barto, Watkins)

  unsupervised, but does not generalize well

- Learning policies with **evolutionary methods** (Koza, Baum)

  policies are general but involve handcrafted features

- Representation of policies using **functional-indexical** representations (Agre, Chapman)

  indexicals like NEXT-NEEDED-BLOCK *hardwired, not learned*

Conclusions

- Approach for learning generalized policies for planning
- Relies on simple concept language for learning useful predicates and rules simultaneously
- Concept languages appear to provide convenient syntax for learning generalized policies in structured domains
- Approach does not make use of handcrafted domain features
- Promising but preliminary results on blocks-world
- Method can hopefully lead to interactive environment for synthesizing domain-dependent control knowledge from declarative domain-descriptions