General and Reusable Indexical Policies and Sketches

Blai Bonet[1] Dominik Drexler[2] Hector Geffner[3,2]

[1]Universitat Pompeu Fabra, Spain
[2]Linköping University, Sweden
[3]RWTH Aachen University, Germany

Introduction

- **Generalized planning** is about finding plans that solve many instances of common domain
- We have shown policies (and sketches) can be expressed by sets of **feature-based rules**
- Rules classify transitions as **good** or **bad** w/o reference to actions: **generality is obtained**
- Rules powerful, effective, and **learnable**
- Also, foundation for GNNs/RL approach

Limitations:

- Policies that enable/disable rule subsets
- No way to "fix attention at given object"
- No principled way to **reuse/compose modules**

Contributions

- **Internal memory states,** that permit to have flow of control to enable/disable rules
- **Indexical features** in terms of **registers**
- **Modules** wrap policies and sketches in units

Example: Delivery

- **Problem:** packages in a grid to delivered
- Two sketches of different "complexity":
  - (Width 2) single rule $\{n>0\} \mapsto \{n\downarrow\}$ where n is num of und. pkgs
  - (Width 0, policy) 4 rules where p and t are distance to **closest** underlivered package and target cell, resp.:

$\{\neg H, p>0\} \mapsto \{p\downarrow, t?\}$	Approach package
$\{\neg H, p=0\} \mapsto \{H\}$	Pick packag
$\{H, t>0\} \mapsto \{t\downarrow\}$	Approach target
$\{H, t=0\} \mapsto \{\neg H, n\downarrow, p?\}$	Deliver package

Problems, Features, and Rules

- Collections $\mathcal{Q}$ of instances over a common planning domain D
- **Sketch** for $\mathcal{Q}$ is set of rules $C \mapsto E$ based on **features** over domain D which can be Boolean or numerical. C contains p and $\neg p$, and $n>0$ and $n=0$, E contains p, $\neg p$, and $p?$, and $n\downarrow$, $n\uparrow$, and $n?$
- State pair (s, s') is **compatible** with $C \mapsto E$ if C satisfied at s, and values change across (s, s') according E: $s' \prec_r s$, and $s' \prec_R s$ if $r \in R$

Serialized Width and Algorithms

- **Width** measures comp. of finding opt. plan. If $w(P) \leq k$, opt. plan in $\mathcal{O}(N^{2k-1})$ with IW(k)
- Sketch R splits P into **subproblems** $P[s]$, like P, but initial state s and goals s' that are either goal of P, or $s' \prec_R s$
- R has **serialized width** $\leq k$ on class $\mathcal{Q}$ if for P in $\mathcal{Q}$, subproblems $P[s]$ have width $\leq k$
- IW(k) is a BFS that **prunes** nodes that don't **discover** k-tuple of atoms. IW(k) runs in $\mathcal{O}(N^{2k-1})$ time, N is number of atoms
- If $w(\mathcal{Q}) \leq k$, IW(k) solves any P in $\mathcal{Q}$ in **ptime**
- IW runs IW(i), $i = 0, 1, 2, \ldots, N$ where N is number of atoms

Algorithm SIW$_R$ for Solving Problems with Sketches

Algorithm: SIW$_R$ search given sketch R

Input: Sketch R over features Φ
Input: Planning problem P with initial state s_0 in which the features in Φ are well defined
$s \leftarrow s_0$
while s is not a goal state of P **do**
  Run IW from s to find s' with s' goal, or $s' \prec_R s$
  if s' is not found **return** UNSOLVABLE
  $s \leftarrow s'$
return path from s_0 to the goal state s

- R is **acyclic** in P if no sequence $s_0, s_1, \ldots, s_n$ such that $s_{i+1} \prec_R s_i$, and $s_n = s_0$
- **Sieve** checks if R is **terminating**. If so R is **acyclic,** and #subp. is polynomial for any P
- If R is terminating and $w(\mathcal{Q}) \leq k$, SIW$_R$ solves any P in $\mathcal{Q}$ in **polynomial time**

General Sketch for $on(x, y)$

Sketch for class $\mathcal{Q}_{on}$ of width 2

Sketch for class $\mathcal{Q}_{on}$ of problems with atomic goal $on(x, y)$ defined with numerical n that counts the number of blocks above x or y, and *On* that represents whether x is on y. Set of features is $\Phi = \{On, n\}$, and rules:

$\{n>0\} \mapsto \{n\downarrow\}$	Put block away from x or y
$\{n=0, \neg On\} \mapsto \{On, n?\}$	Stack x on y

Memory, Registers, and Indexicals

- Sketch with memory is $\langle M, \Phi, m_0, R\rangle$ where M is memory states, Φ is features, m_0 is initial memory, and R is rules $(m, C) \mapsto (E, m')$ where $C \mapsto E$ is standard rule, and m and m' are memory states
- Registers store objects, that can be referred in features that become indexical
- Registers $\mathfrak{R} = \{\mathfrak{r}_0, \mathfrak{r}_1, \ldots\}$ contain objects selected with using new *concepts* C and *roles* R
- **Indexical feature** is function of state and registers; like "dist. to object stored in $\mathfrak{r}_0$"
- Effects *Load*(C, $\mathfrak{r}$) to update $\mathfrak{r}$. Rule with *Load*(C, $\mathfrak{r}$) has condition C >0, single load effect, and also ϕ? for all ϕ in $\Phi(\mathfrak{r})$
- Rules with loads called *internal*, other *external*

Clearing Multiple Blocks

Indexical policy for the class $\mathcal{Q}_{clear^*}$

Class $\mathcal{Q}_{clear^*}$ of problems whose goal is conjunction of $clear(x)$ atoms. Concept C contains blocks to be cleared

Policy π_{clear^*} with 5 memory states, 2 registers, concept N for blocks in C that are not clear, indexical T for topmost block above $\mathfrak{r}_0$, Boolean H iff some held, and Boolean A iff block in $\mathfrak{r}_1$ is above some in C.

$m_0 \,\|\, \{H\} \mapsto \{\neg H, \mathsf{N}?\} \,\|\, m_1$
$m_0 \,\|\, \{\neg H\} \mapsto \{\} \,\|\, m_1$

$m_1 \,\|\, \{\mathsf{N}>0\} \mapsto \{Load(\mathsf{N}, \mathfrak{r}_0), \mathsf{T}?\} \,\|\, m_2$
$m_2 \,\|\, \{\mathsf{T}>0\} \mapsto \{Load(\mathsf{T}, \mathfrak{r}_1), A?\} \,\|\, m_3$
$m_2 \,\|\, \{\mathsf{T}=0\} \mapsto \{\} \,\|\, m_1$

$m_3 \,\|\, \{\neg H, A\} \mapsto \{H, \neg A, \mathsf{N}?, \mathsf{T}?\} \,\|\, m_4$
$m_4 \,\|\, \{H\} \mapsto \{\neg H\} \,\|\, m_2$
$m_4 \,\|\, \{H\} \mapsto \{\neg H, \mathsf{N}\downarrow\} \,\|\, m_2$

Reusable Modules

- Module is $\langle$args, $Z, M, \mathfrak{R}, \Phi, m_0, R\rangle$ where args $= \langle x_1, x_2, \ldots, x_n\rangle$ is arguments, Z and Φ are features, M is memory, $\mathfrak{R}$ is registers, $m_0 \in M$ is initial memory, and R is rules
- Features in Φ used in rules may depend on arguments, registers, and features in Z
- New **call and do rules** to call other modules, and execute ground actions
- Call/do rules $(m, C) \mapsto (\mathtt{name}(v_1, \ldots, v_n), m')$ where m and m' are memory, C is condition, `name` is module or action schema name, and each value v_i is of appropriate type
- If *call rule*, sketch for module `name` executed until no rules applicable, and control returned to m'. If *do rule*, apply ground action $\mathtt{name}(o_1, \ldots, o_n)$, where o_i belongs to v_i, and control returned at m'

Solving Blocksworld Problems

Module `tower`(O, X) for building a single tower

Aimed at class $\mathcal{Q}_{tower}$ of problems where blocks to be stacked in *single tower* achieving $\wedge_{i=1}^{k} on(x_i, x_{i-1})$ and $ontable(x_0)$. Module is $\langle\langle \mathsf{O}, \mathsf{X}\rangle, Z, M, \mathfrak{R}, \Phi, m_0, R\rangle$ where O is role argument that contains pairs $\{(x_i, x_{i-1}) \mid i = 1, \ldots, k\}$, and X is concept with lowest misplaced block in target tower

Other elements are $Z = \emptyset$, $M = \{m_0, m_1, \ldots, m_3\}$, $\mathfrak{R} = \{\mathfrak{r}_0\}$, and $\Phi = \{\mathsf{M}, \mathsf{W}\}$ where M is indexical that contains block to be placed above $\mathfrak{r}_0$ according to O, if any, and, W contains block directly below $\mathfrak{r}_0$, if any, according to the target tower O.

*% Module **tower**(O, X)*
$m_0 \,\|\, \{\mathsf{X}>0\} \mapsto \{Load(\mathsf{X}, \mathfrak{r}_0), \mathsf{M}?, \mathsf{W}?\} \,\|\, m_1$
$m_1 \,\|\, \{\mathsf{W}=0\} \mapsto \mathtt{on\text{-}table}(\mathfrak{r}_0) \,\|\, m_2$
$m_1 \,\|\, \{\mathsf{W}>0\} \mapsto \mathtt{on}(\mathfrak{r}_0, \mathsf{W}) \,\|\, m_2$
$m_2 \,\|\, \{\mathsf{M}>0\} \mapsto \mathtt{tower}(\mathsf{O}, \mathsf{M}) \,\|\, m_3$

Module `blocks`(O) for arbitrary towers

Aimed at class $\mathcal{Q}_{blocks}$ of problems for building many target towers, takes single role argument O. Module is tuple $\langle\langle \mathsf{O}\rangle, Z, M, \mathfrak{R}, \Phi, m_0, R\rangle$ where $Z = \emptyset$, $M = \{m_0, m_1\}$, $\mathfrak{R} = \{r_0\}$, and $\Phi = \{\mathsf{L}\}$ where L is contains the lowest misplaced blocks in O

*% Module **blocks**(O)*
$m_0 \,\|\, \{\mathsf{L}>0\} \mapsto \{Load(\mathsf{L}, \mathfrak{r}_0)\} \,\|\, m_1$
$m_1 \,\|\, \{\} \mapsto \mathtt{tower}(\mathsf{O}, \mathfrak{r}_0) \,\|\, m_0$

Conclusions and Future Work

- Extensions to make policies and sketches more expressive and reusable: (1) internal memory states, (2) indexical concepts and features, and (3) modules that wrap up policies and sketches
- **Future work:** learn policies and sketches bottom-up, theoretical properties, etc