Discovering Options from Example Trajectories

Peng Zang
Peng Zhou
David Minnen
Charles Isbell
If we had a few expert traces, trajectories of the problem being solved

We can find problem decompositions by examining those trajectories

And solve the problem faster
Taxi Domain

State features:
X – taxi X
Y – taxi Y
P – passloc
D – destination

Actions:
N – North
S – South
E – East
W – West
P – Pickup
D – Dropoff
Problem Decomposition

- Taxi task
  - pickup passenger
    - goto(passloc)
    - pickup
  - dropoff passenger
    - goto(dest)
    - dropoff
Problem Decomposition

- Recognition

  North, north, west, west, west, pickup, south, south, south, south, dropoff

  goto( ), pickup, goto( ), dropoff

  pickup-passenger, dropoff-passenger
Problem Decomposition

- **Transfer**
  - Skills applicable to other tasks
    
    goto( ), goto( )

  - Decomposition applicable to other action sets
    
    pickup-passenger, dropoff-passenger
Problem Decomposition

goto(□)

**State features:**
X – taxi X
Y – taxi Y

**Actions:**
N – North
S – South
E – East
W – West

pickup-passenger

**State features:**
X – taxi X
Y – taxi Y
P – passloc

**Actions:**
R – goto(■)
G – goto(■)
Y – goto(■)
B – goto(■)
P – Pickup

- **Speedup**
  - Breaks up the problem
  - Abstraction opportunities in subproblems
  - Reuse
Focus of this work:

Automating decomposition by finding and factoring out subtasks

- Action sequences contain signatures use to find good subtasks
- Heuristic for finding good subtask boundaries: “abstraction boundaries”
- Direct incorporation of options solve transition and reward model
Definitions

- **SMDP**: (S, A, P, R, γ)
  - feature based representation
  - known transition/reward model and action feature dependencies
- **Subproblem**: (M, F, A, ω)
- **Option**: (I, π, β)
- **Trajectory**: (s,a,s',d,r) sequence

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>NextState</th>
<th>Duration</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>X:0 Y:0</td>
<td>East</td>
<td>X:1 Y:0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>X:1 Y:0</td>
<td>East</td>
<td>X:2 Y:0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>X:2 Y:0</td>
<td>Airport</td>
<td>X:9 Y:8</td>
<td>20</td>
<td>-15</td>
</tr>
</tbody>
</table>
Automated Decomposition
(What are good subproblems?)

- Size: large enough to capture a significant portion of the problem but not the whole thing
- Frequency: reuse opportunities
- Abstraction: offers significant speedups
Automated Decomposition

- **Observation:** Subproblems of significant size and frequency leave long, common action sequences in the trajectories that act as “signatures”
- **Intuition:** We can use these “signatures” to find good subproblems

- **Observation:** Different subproblems require different state features
- **Intuition:** Use “abstraction breaks” to find subtask boundaries.
Discovery Algorithm

- Suffix tree to find common action sequences
- Extend to find goals
- Choose subtask based on most frequent goal
- Find and extract all instances of subtask
- Repeat
Automated Decomposition

**Trajectories**

WWWNNNPSSSSSD
EESSPWWWNNWWND
NNEPSSSSSD
NNWWWPSESSSEEED
WPNNNNND
WPSSSEEESED
...

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Automated Decomposition

Trajectories
WWW NNPSSSSSD
EESSPWWW NNN WND
NN EPSSSSSD
NN WWWPSESSSEEEED
WP NNNN D
WPSSSEEESED
...

Find common action sequences using suffix tree
Automated Decomposition

Trajectories
WWWNNPSSSSD
EESSPWWWNNNWND
NNEPSSSSD
NNWWWPSESSSEEED
WPNNNND
WPSSSEEESED
...

we are navigating, only worrying about taxi location (X,Y). Now with the Pickup action we also need to worry about passenger location. Let's make this a boundary.

Extend to find goals.

We stop when the abstraction suddenly changes. Intuition is that this denotes a conceptual boundary. It also maximizes speedup opportunities.
Automated Decomposition

Most common goal selected.
This creates subproblem:
M: Taxi SMDP
F: X, Y
A: N,W
ω: X:0 Y:0
Automated Decomposition

<table>
<thead>
<tr>
<th>P</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Extract subproblem:

\[ \alpha \rightarrow \text{Goto (X:0 Y:0)} \]

Note: we estimate the speed up for any proposed subproblem. We only perform extraction if we expect it to offer speedup

<table>
<thead>
<tr>
<th>Trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWWN</td>
</tr>
<tr>
<td>EESSP</td>
</tr>
<tr>
<td>NNEPSSSSSD</td>
</tr>
<tr>
<td>NNWWW</td>
</tr>
<tr>
<td>WP</td>
</tr>
<tr>
<td>W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Trajectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )PSSSSSD</td>
</tr>
<tr>
<td>EESSP</td>
</tr>
<tr>
<td>NNEPSSSSSD</td>
</tr>
<tr>
<td>( \alpha )PSESSSSEED</td>
</tr>
<tr>
<td>WP</td>
</tr>
<tr>
<td>( \alpha )PSSSEEESED</td>
</tr>
</tbody>
</table>

...
Automated Decomposition

Trajectories
αPSSSSSD
EESSPαD
NNEPSSSSSD
αPSESSSEEEED
WPαD
αPSSSEEESED
...

Repeat finding additional subproblems until no subproblems are found. Subproblems are solved and inserted into the base problem. When no more subproblems can be found, we solve the augmented MDP.
Results

- Two Domains
  - Taxi World
  - Wargus (Simplified)
Taxi World

State features:
- X – taxi X
- Y – taxi Y
- P – Pickup location
- D – Destination

Actions:
- N – North
- S – South
- E – East
- W – West
- P – Pickup
- D – Dropoff

- Classic RL Problem
- Easily scalable, good for testing
- Deterministic and stochastic versions
Wargus (Simplified RTS)

- Strategic planning problem
- Focuses on learning build order for grunt rush
- More features

**Features:**
- Gold (hundreds)
- Wood (hundreds)
- Grunts
- Farms
- LumberMill
- Barracks
- Blacksmith
- Time-of-day
- Location
- Status

**Actions:**
- No-op
-_gotoGoldmine
- GotoWoods
- GotoTownhall
- Chop
- Mine
- Deposit
- BuildFarm
- BuildBarracks
- BuildLumbermill
- BuildBlacksmith
- TrainGrunt
Robustness

Number of expert trajectories

Quality of expert trajectories
Speedup

One passenger

Two passengers
Discovered Options (Taxi)

- Goto
- PickUp and Goto
Discovered Options (Wargus)

Low level options

Mine gold
(goto-goldmine, mine, deposit, etc.)

Chop wood
(goto-woods, chop, deposit, etc.)

High level options

Build a lumbermill
(chopwood, build-lumbermill, etc.)

Build a farm
(minegold, build-farm, etc.)
Conclusion

- Use expert traces for problem decomposition
- Requires
  - Feature based state representation
  - Known transition/reward model and action feature dependencies
  - Some expert traces
- Buys you
  - Problem decomposition
  - Speedup