# Discovering Options from Example Trajectories

Peng Zang
Peng Zhou
David Minnen
Charles Isbell

The Laboratory for Interactive Artificial Intelligence

Georgia Tech College of Computing
School of Interactive Computing

If we had a few expert traces,

trajectories of the problem being solved

We can find problem decompositions

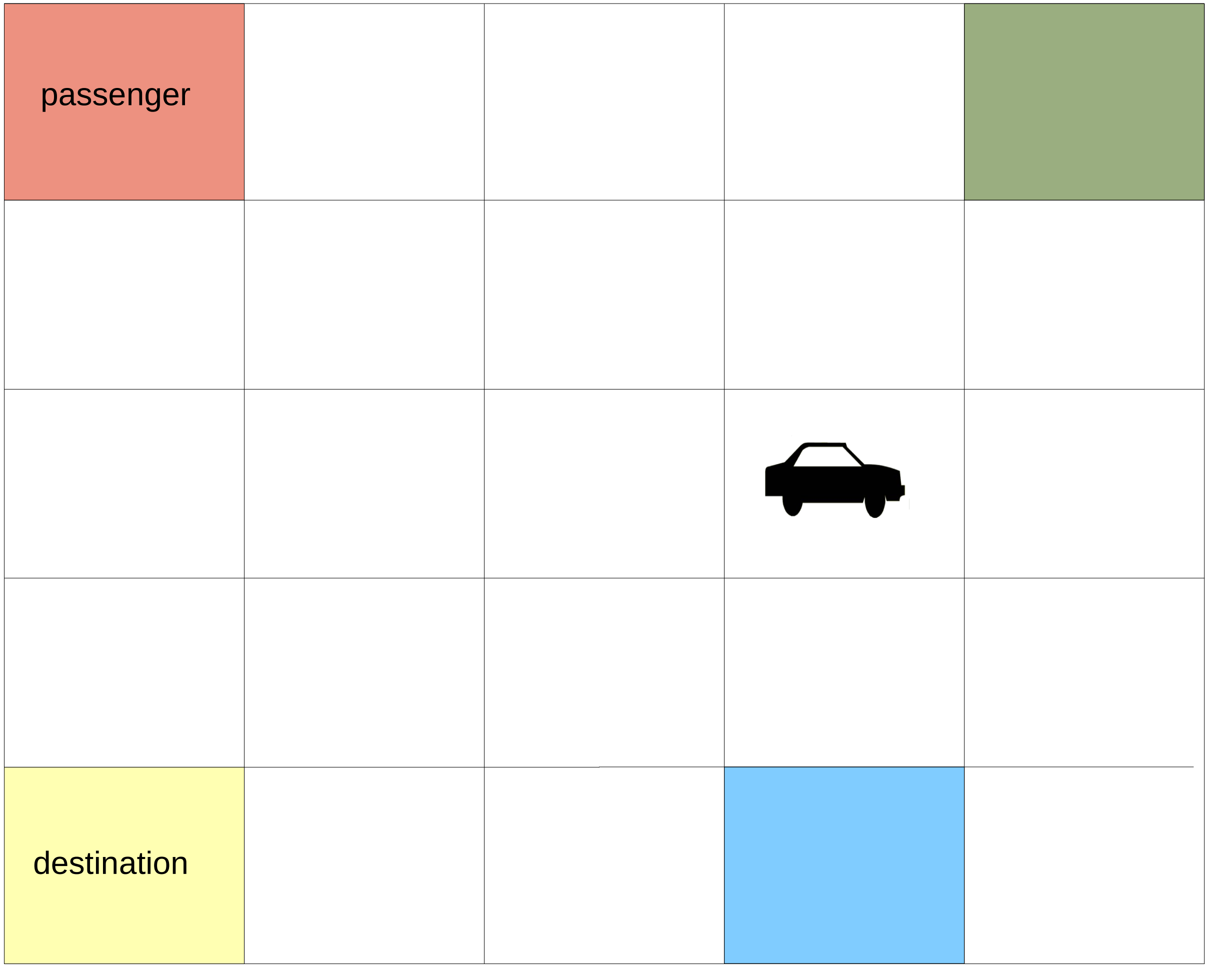
by examining those trajectores

And solve the problem faster

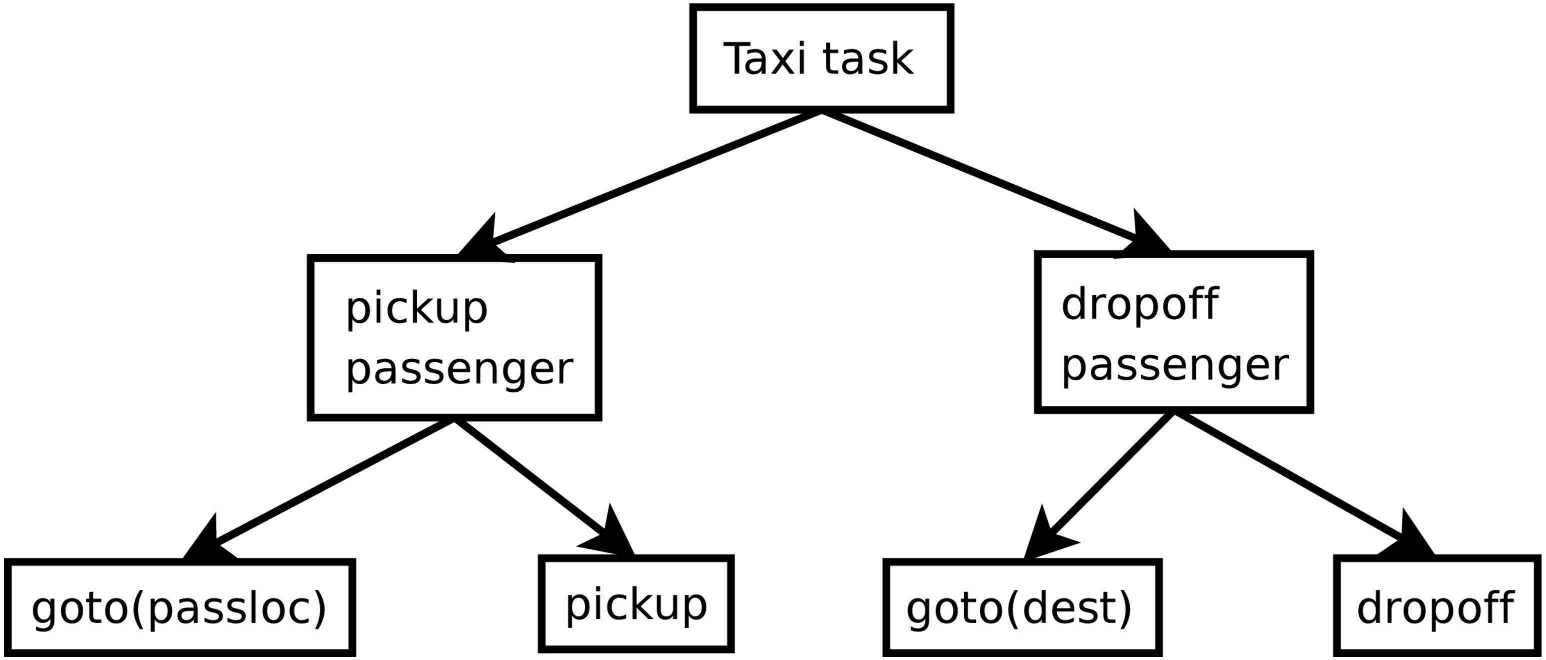
# Taxi Domain

**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



# 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

Finding decomposition

- Action sequences contain signatures

  use to find good subtasks

- Heuristic for finding good subtask boundaries:

  “abstraction boundaries”

Using decomposition

- 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

| **State** | **Action** | **NextState** | **Duration** | **Reward** |
|---|---|---|---|---|
| X:0 Y:0 | East | X:1 Y:0 | 1 | -1 |
| X:1 Y:0 | East | X:2 Y:0 | 1 | -1 |
| X:2 Y:0 | Airport | X:9 Y:8 | 20 | -15 |
| … | | | | |

# 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

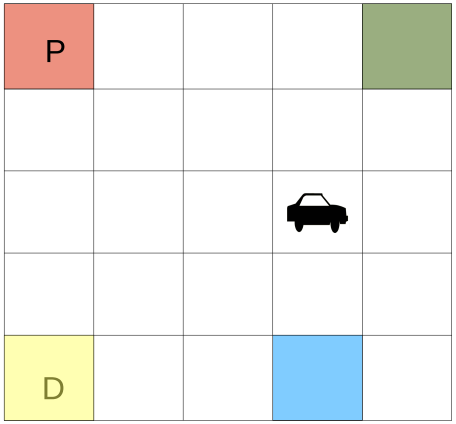
- Observation: Subproblems of significant size and frequency leave long, common action sequences in the trajectories that act as “signatures”
- Intuition: We can use these “signatures” to find good subproblems

- 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

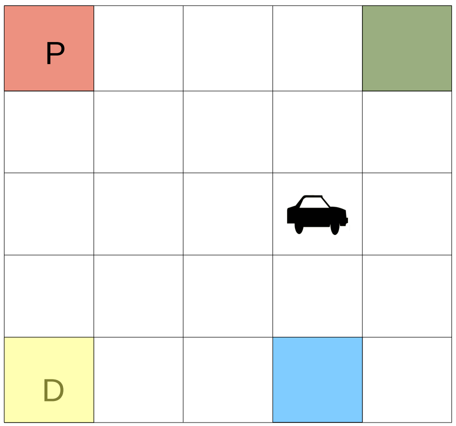
# Automated Decomposition




**Trajectories**

WWWNNPSSSSD
EESSPWWWNNNWND
NNEPSSSSD
NNWWWPSESSSEEED
WPNNNND
WPSSSEEESED
...

# Automated Decomposition




**Trajectories**

WWWNNPSSSSD

EESSPWWWNNNWND

NNEPSSSSD

NNWWWPSESSSEEED

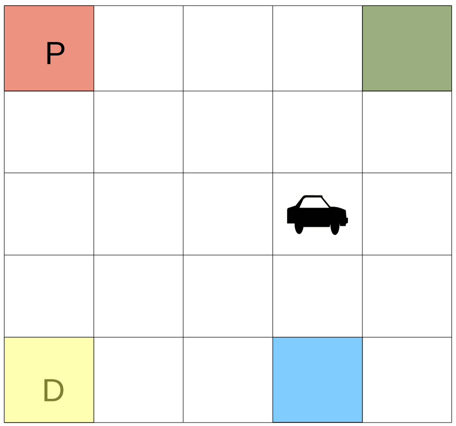
WPNNNND

WPSSSEEESED

...

**Find common action sequences using suffix tree**

# Automated Decomposition




X:0 Y:0

X:4 Y:0

**Trajectories**

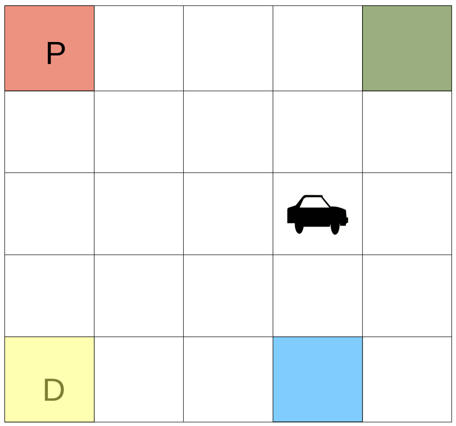
WWWNNPSSSSD
EESSPWWWNNNWND
NNEPSSSSD
NNWWWPSESSSEEED
WPNNNND
WPSSSEEESED
…

**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




**Trajectories**

WWWNNPSSSSD
EESSPWWWNNNWND
NNEPSSSSD
NNWWWPSESSSEEED
WPNNNND
WPSSSEEESED
...

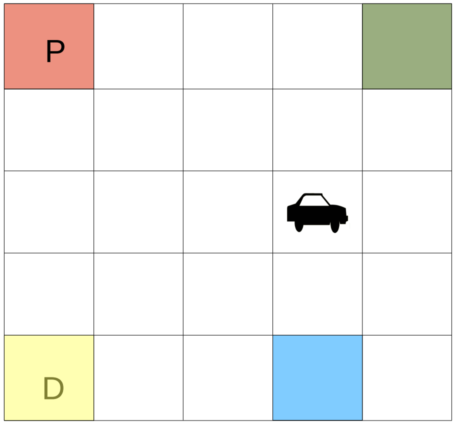
X:4 Y:0

**X:0 Y:0**

**Most common goal selected.**
**This creates subproblem:**

- **M: Taxi SMDP**
- **F: X, Y**
- **A: N,W**
- **ω: X:0 Y:0**

# Automated Decomposition




**Trajectories**

WWWNNPSSSSD
EESSPWWWNNNWND
NNEPSSSSD
NNWWWPSESSSEEED
WPNNNND
WPSSSEEESED
...

Extract subproblem:

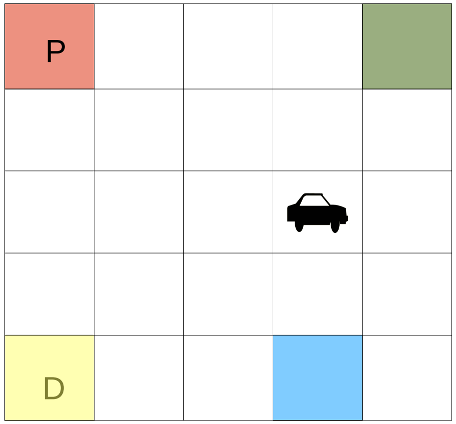
α – Goto (X:0 Y:0)

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

**New Trajectories**

αPSSSSD
EESSPαD
NNEPSSSSD
αPSESSSEEED
WPαD
αPSSSEEESED
...

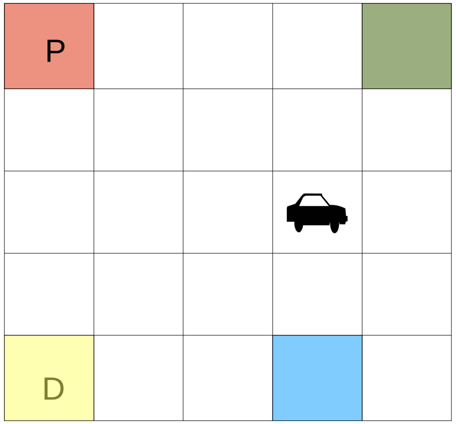
# Automated Decomposition




**Trajectories**
αPSSSSD
EESSPαD
NNEPSSSSD
αPSESSSEEED
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.**

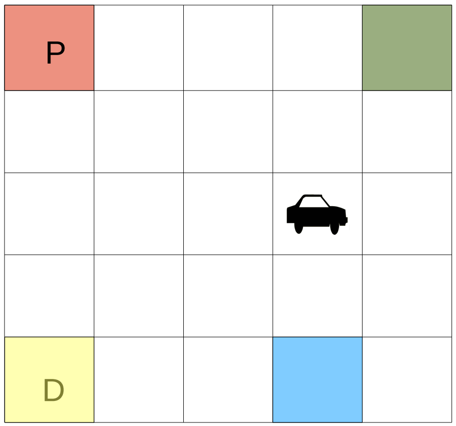
# Results





- 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)


Features:

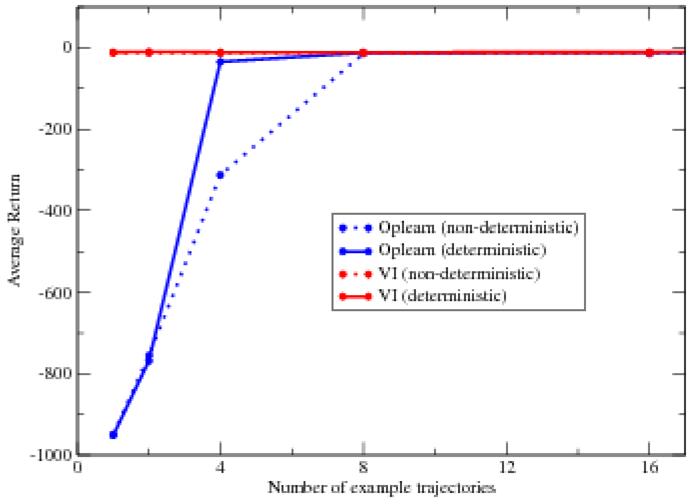
- Gold (hundreds)
- Wood (hundres)
- Grunts
- Farms
- LumberMill
- Barracks
- Blacksmith
- Time-of-day
- Location
- Status

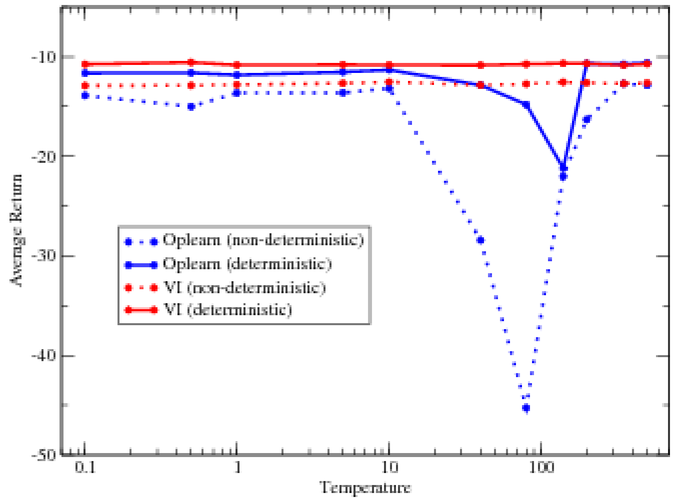
Actions:

- No-op
- GotoGoldmine
- GotoWoods
- GotoTownhall
- Chop
- Mine
- Deposit
- BuildFarm
- BuildBarracks
- BuildLumbermill
- BuildBlacksmith
- TrainGrunt

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

# Robustness

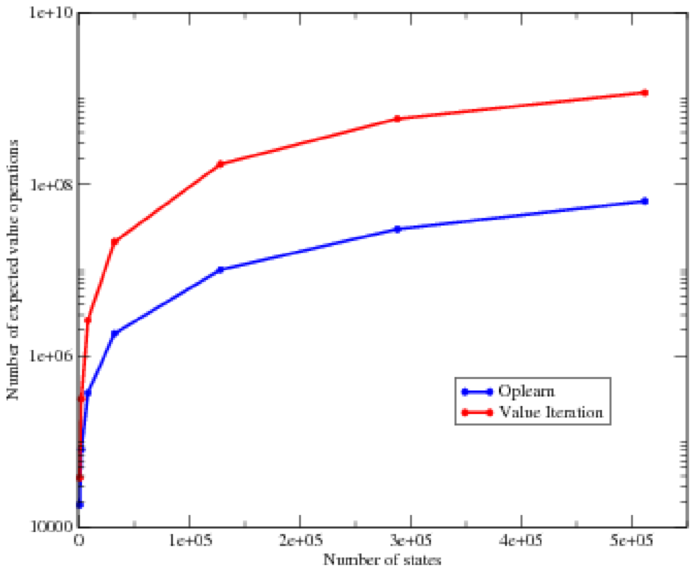


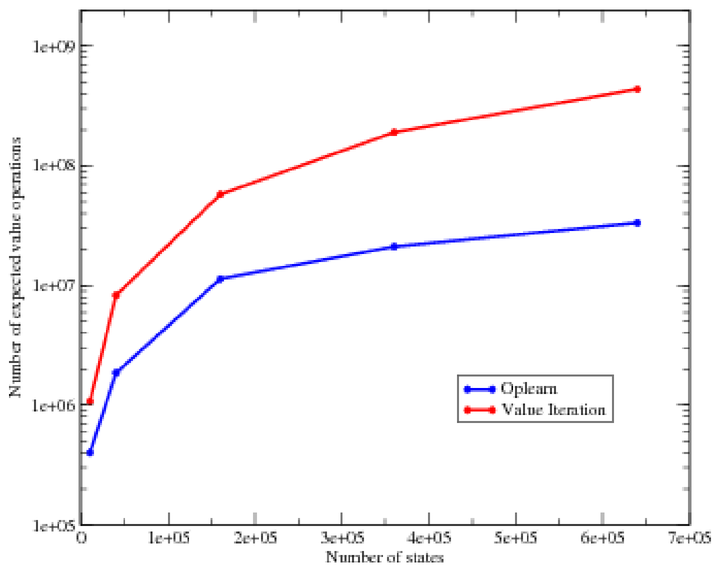

Number of expert trajectories

Quality of expert trajectories

# Speedup

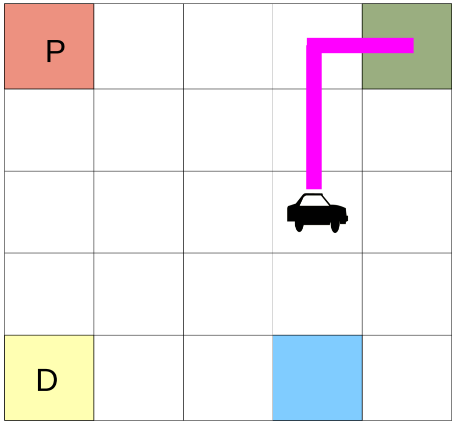


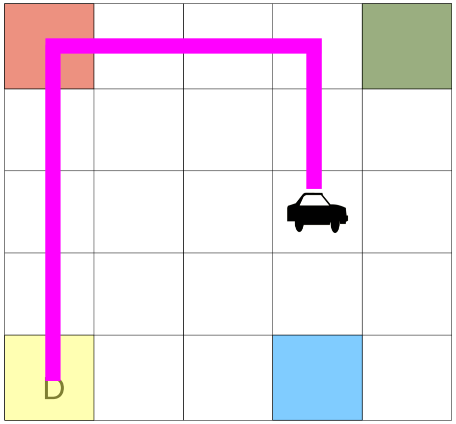

One passenger




Two passengers

# Discovered Options (Taxi)



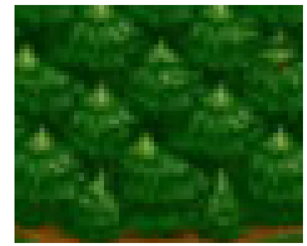




- Goto
- PickUp and Goto

# Discovered Options (Wargus)

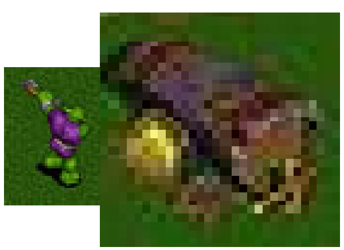
## Low level options

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