Problems of RL

Curse of Dimensionality
In real world problems ist difficult/impossible to define discrete state-action spaces.

(Temporal) Credit Assignment Problem
RL cannot handle large state action spaces as the reward gets too much dilited along the way.

Partial Observability Problem
In a real-world scenario an RL-agent will often not know exactly in what state it will end up after performing an action. Furthermore states must be history independent.

State-Action Space Tiling
Deciding about the actual state- and action-space tiling is difficult as it is often critical for the convergence of RL-methods. Alternatively one could employ a continuous version of RL, but these methods are equally difficult to handle.

Non-Stationary Environments
As for other learning methods, RL will only work quasi stationary environments.

Real-world behavior is hierarchical

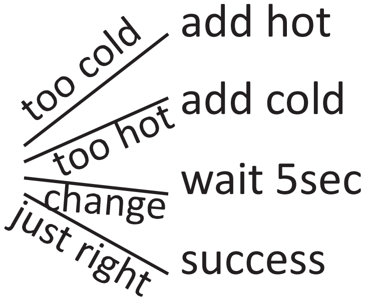
1. pour coffee

2. add sugar

3. add milk

4. stir

1. set water temp
2. get wet
3. shampoo
4. soap
5. turn off water
6. dry off



Hierarchical Reinforcement Learning

- Exploits domain structure to facilitate learning
  - Policy constraints
  - State abstraction
- Paradigms: Options, HAMs, MaxQ, etc.

Advantages of HRL

1. Faster learning
   (mitigates scaling problem)
2. Structured exploration
   (explore with sub-policies rather than primitive actions)
3. Transfer of knowledge from previous tasks
   (generalization, shaping)

Semi-Markov Decision Process

- Generalizes MDPs
- Action **a** takes **N** steps to complete in **s**
- P(**s’**,**n** | **a**, **s**), R(**s’**, **N** | **a**, **s**)
- Bellman equation:

$$V^{\pi}(s) = \sum_{s',N} P(s',N|s,\pi(s)) \left[R(s',N|s,\pi(s)) + \gamma^{N} V^{\pi}(s')\right] .$$

$$V^{\pi}(s) = \overline{R}(s,\pi(s)) + \sum_{s',N} P(s',N|s,\pi(s))\gamma^{N} V^{\pi}(s').$$

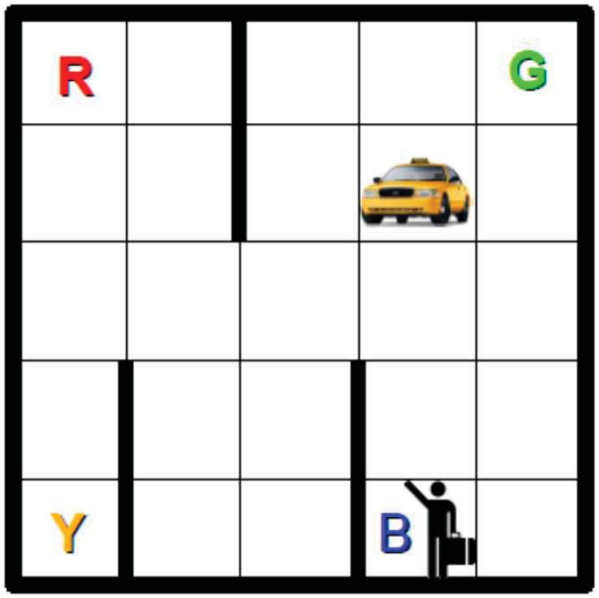
Semi-Markov Decision Process

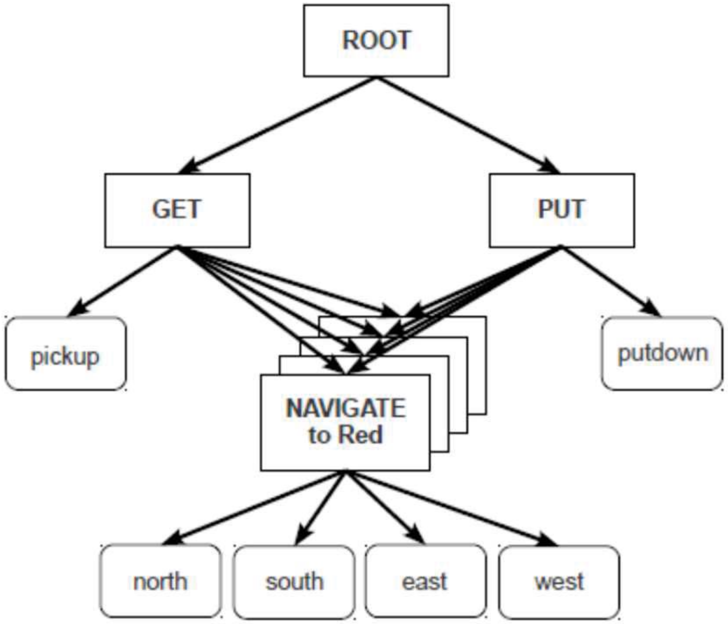
- Generalizes MDPs
- Action **a** takes **N** steps to complete in **s**
- P(**s’**,**n** | **a**, **s**), R(**s’**, **N** | **a**, **s**)
- Bellman equation:

$$V^*(s) = \max_a \left[R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) V^*(s') \right]$$

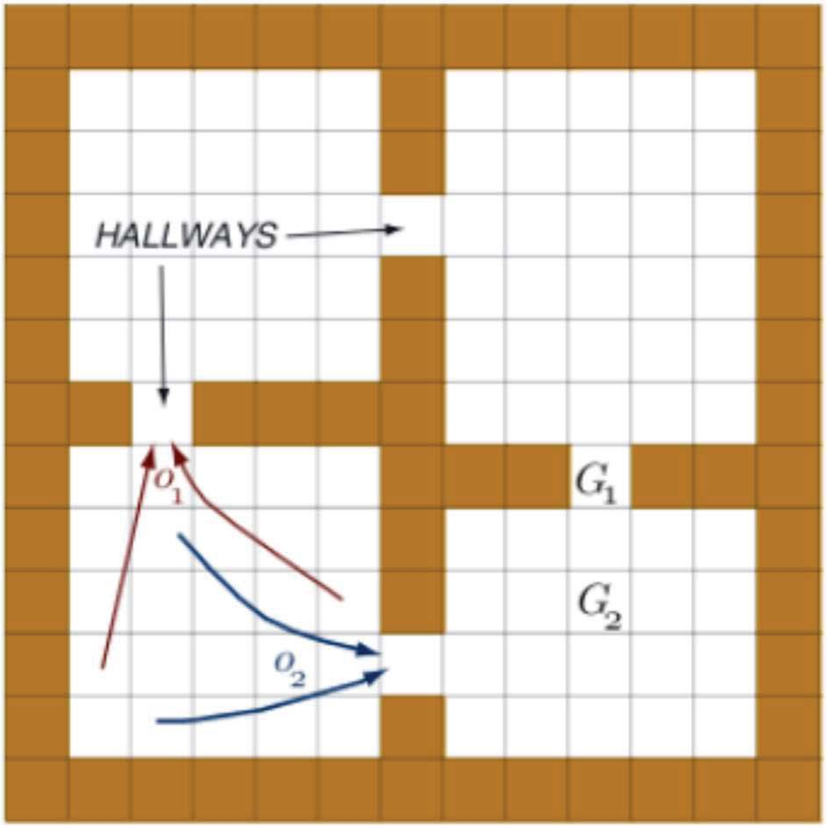
$$Q^*(s,a) = R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) \max_b Q^*(s',b)$$

Taxi Example

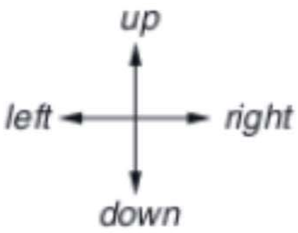




Room Example



4 stochastic primitive actions



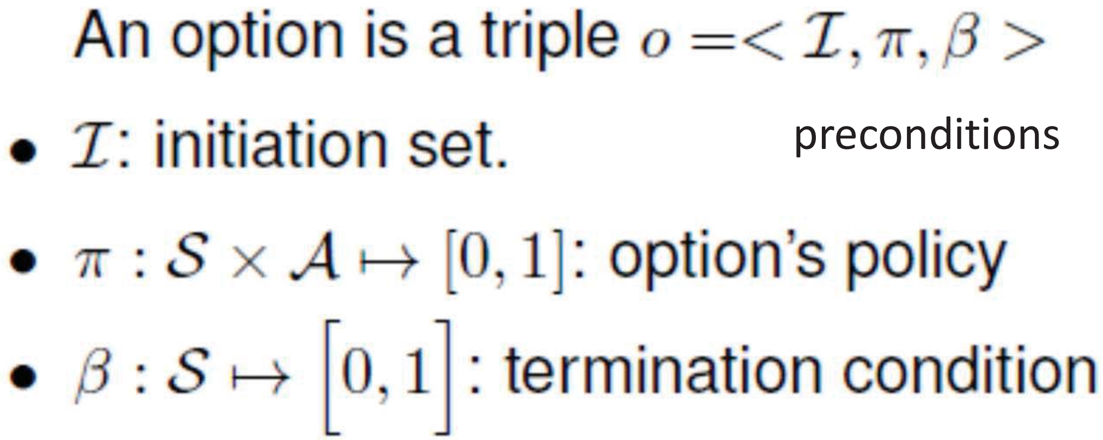
Fail 33% of the time

8 multi-step options

(to each room's 2 hallways)

Sutton, Precup, Singh, 1999

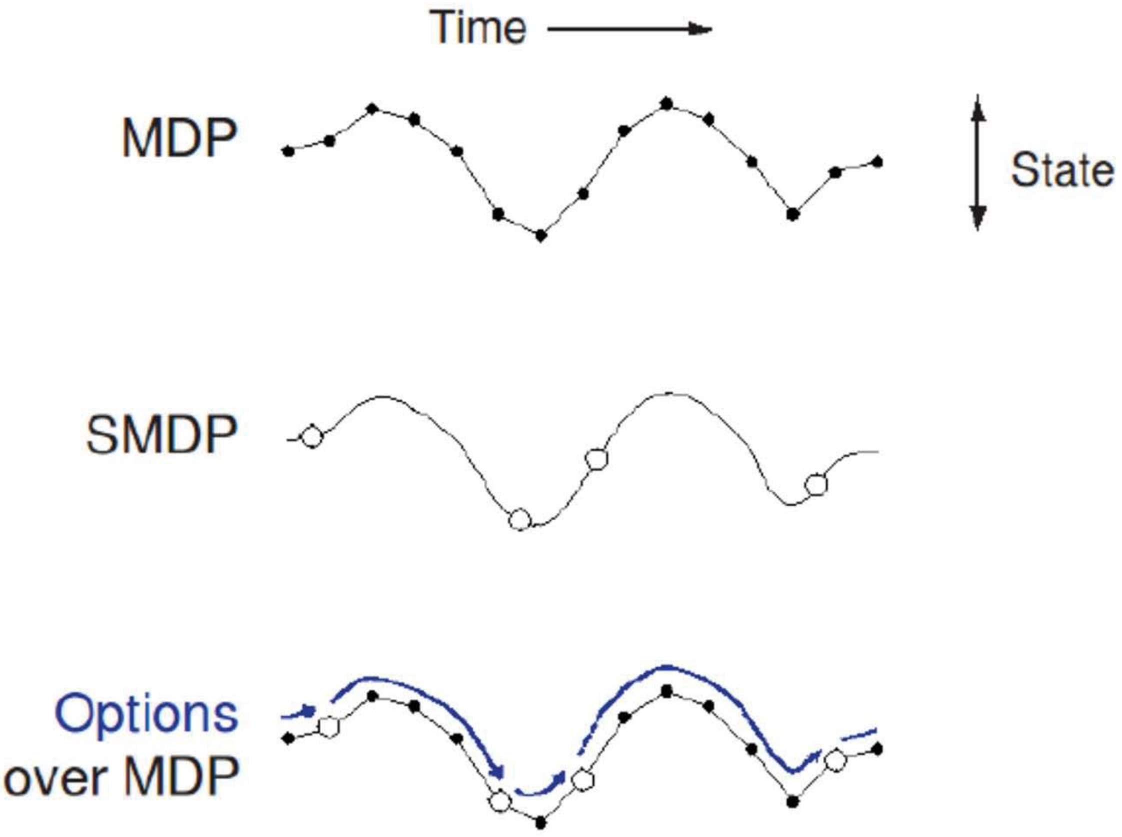
Options



An option is a triple $o = < \mathcal{I}, \pi, \beta >$

- $\mathcal{I}$: initiation set. preconditions
- $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$: option's policy behavior
- $\beta : \mathcal{S} \mapsto \left[0,1\right]$: termination condition effect

Options



Options

option's policy: π_i; global policy: μ

- reward part of option:

$$r(s,o) = \mathsf{E}\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^k r_{t+k} | o, s_t = s \right\}$$

- prediction-state part:

$$p(s'|s,o) = \sum_{k=1}^{\infty} p(s',k|s,o)\gamma^k$$

policy over options $\mu : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$

$$\begin{aligned} V^{\mu}(s) &= \mathsf{E}\left\{ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} .... | \mu, s_t = s \right\} \\ &= \mathsf{E}\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k}) | \mu, s_t = s \right\} \\ &= \mathsf{E}\left[r(s,o) + \sum_{s_{t+k}} p(s_{t+k}|s,o) V^{\mu}(s') | \mu, s_t = s \right] \end{aligned}$$

Options

option's policy: π_i; global policy: μ

- reward part of option:

$$r(s,o) = \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^k r_{t+k} | o, s_t = s\Big\}$$

- prediction-state part:

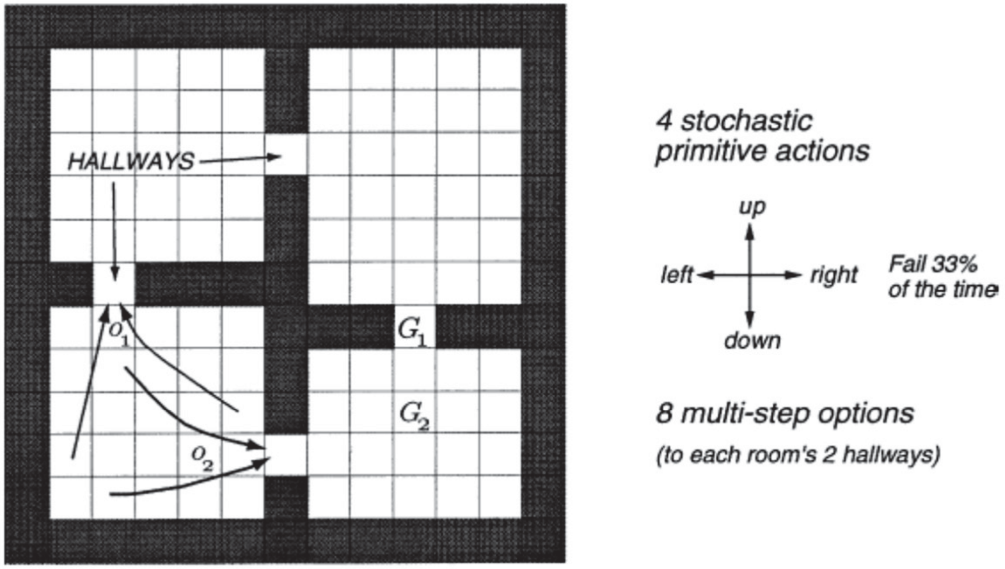
$$p(s'|s,o) = \sum_{k=1}^{\infty} p(s', k|s,o)\gamma^k$$

policy over options $\mu : \mathcal{S} \times \mathcal{O} \to [0, 1]$

$$\begin{aligned} Q^{\mu}(s,o) &= \mathsf{E}\Big\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} .... | o\mu, s_t = s\Big\} \\ &= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k}) | \mu, s_t = s\Big\} \\ &= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} \\ &+ \max_{o'} \mu(s_{t+k}, o') Q^{\mu}(s_{t+k}, o') | o\mu, s_t = s\Big\} \\ &= \mathsf{E}\Big\{r(s,o) + \sum p(s_{t+k}|s,o) \max_{o'} \mu(s_{t+k}, o') Q^{\mu}(s_{t+k}, o')\Big\} \end{aligned}$$

Options

Gridworld environment with stochastic cell-to-cell actions and room-to-room hallway options. Two of the hallway options are suggested by the arrows o1 and o2. G1 and G2 are goals



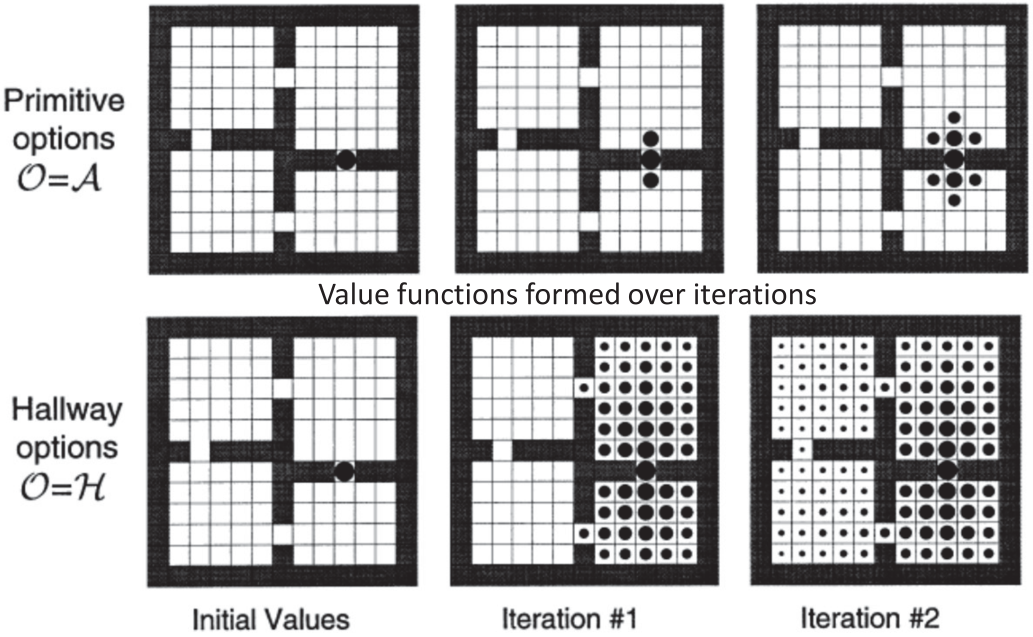
4 stochastic primitive actions

Fail 33% of the time

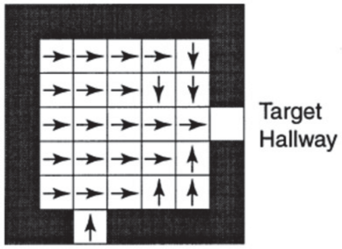
8 multi-step options

(to each room's 2 hallways)

hallway options take the agent from anywhere within the room to one of the two hallway cells leading out of the room



Value functions formed over iterations

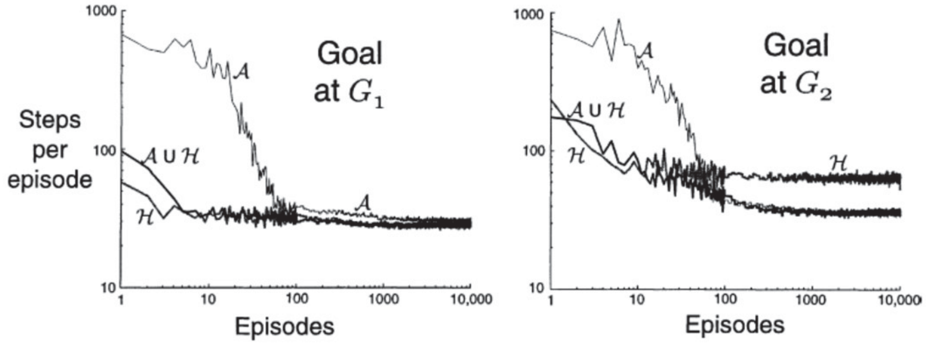


Options

SMDP Q-learning: given the set of defined options.
– execute the current selected option (e.g use epsilon greedy $Q(s, o)$) to termination.
– compute $r(s_t, o)$, then update $Q(s_t, o)$ as Q-learning/SARSA.

$$Q(s,o) \leftarrow Q(s,o) + \alpha\left[r + \gamma^k \max_{o' \in \mathcal{O}_{s'}} Q(s',o') \quad Q(s,o)\right]$$

If primitive actions are included as options, then optimal with options is like optimal without options



Options

SMDP Q-learning: given the set of defined options.

– execute the current selected option (e.g use epsilon greedy $Q(s,o)$) to termination.

– compute $r(s_t,o)$, then update $Q(s_t,o)$ as Q-learning/SARSA.

Intra-option Q-learning: partially defined options

– after each primitive action, update all the options (off-policy learning).

– converge to correct values, ”under same assumptions as 1-step Q-learning” (Sutton)

$$Q_{k+1}(s_t,o) = (1-\alpha_k)Q_k(s_t,o) + \alpha_k\left[r_{t+1} + \gamma U_k(s_t,o)\right]$$

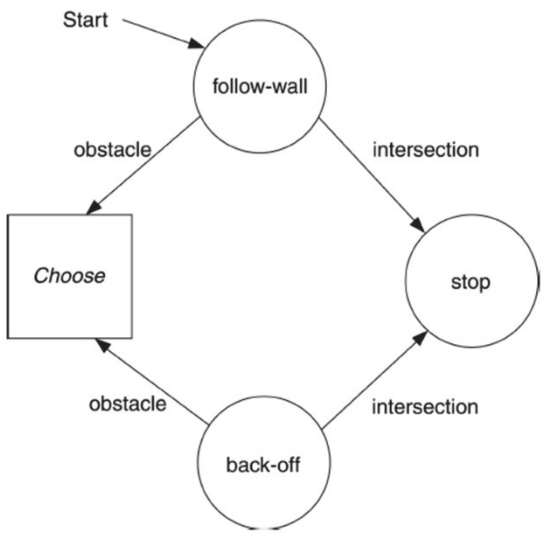
where

$$U_k(s,o) = (1-\beta(s))Q_k(s,o) + \beta(s)\max_{o' \in \mathcal{O}} Q_k(s,o')$$

Hierarchies of Abstract Machines

Partially specified Programs [Parr Russell 98]

- MDP State and Machine State



$$Q_{k+1}([s_c, m_c], a_c) = (1 - \alpha_k) Q_k([s_c, m_c], a_c) + \alpha_k [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{\tau-1} r_{t+\tau} + \gamma^{\tau} \max_{a'} Q_k([s'_c, m'_c], a')]$$

Task Hierarchy

MAXQ Task hierarchy [Dietterich 2000]

- Directed acyclic graph of subtasks
- Hierarchy of SMDS to be simultaneously learned
- Leaves are the primitive MDP actions

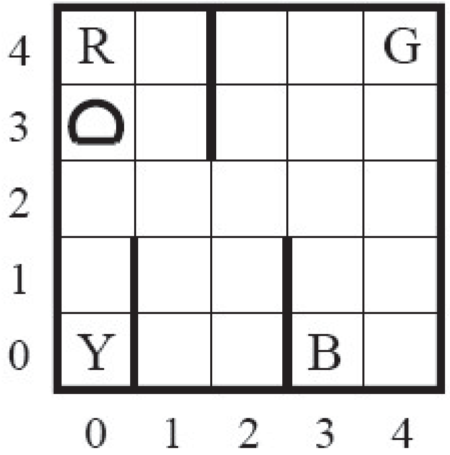
Traditionally, task structure is provided as prior knowledge to the learning agent

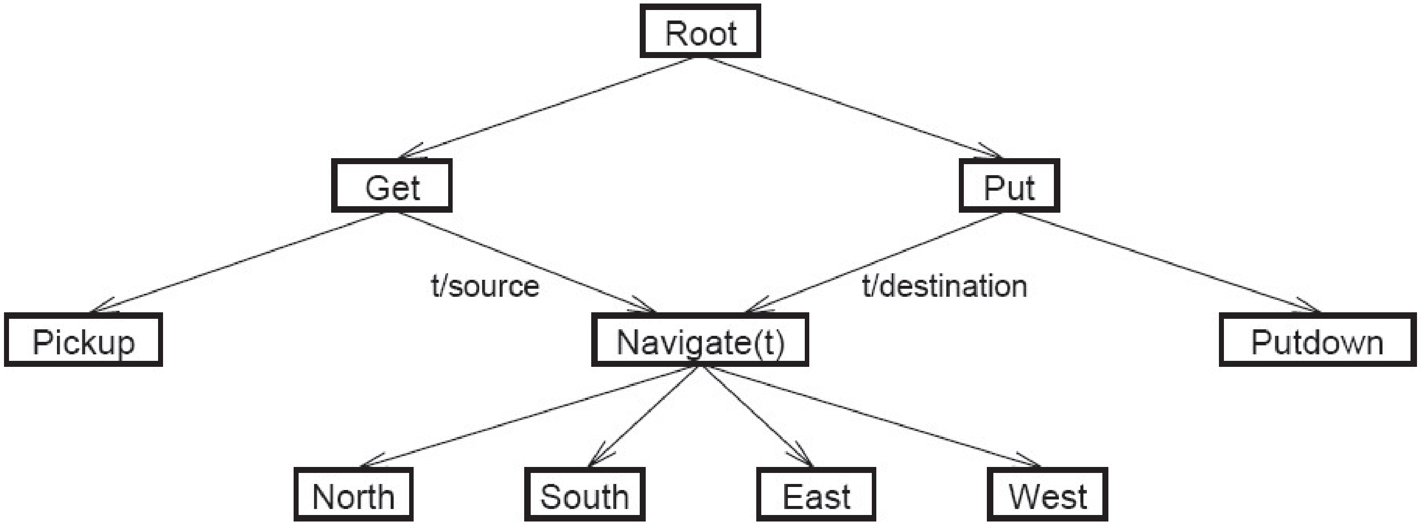
Each task associated with termination, Actions, and pseudo reward function: $\langle T_i, A_i, R_i \rangle$

Hierarchical policy is a set of policies, one for each subtask

Taxi Domain

- Motivational Example
- Reward: -1 actions,
  -10 illegal, 20 mission.
- 500 states
- Task Graph:





MAXQ Alg. (Value Fun. Decomposition)

- Compactness) in the representation of the hierarchical value function
- Re-write Q(p, s, a) as

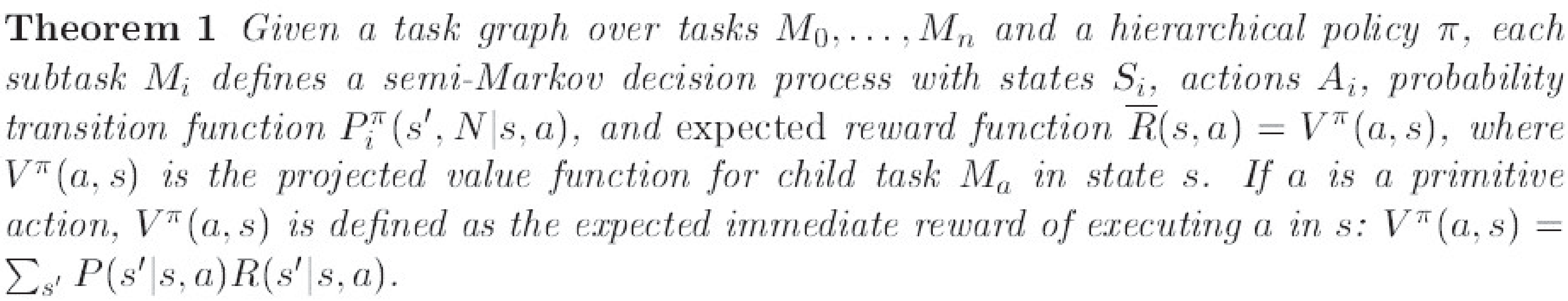
$$Q(p, s, a) = V(a, s) + C(p, s, a)$$

$$V(p, s) = \max_a [V(a, s) + C(p, s, a)]$$

where *V(a, s)* is the expected total reward while executing action a, and *C(p, s, a)* is the expected reward of completing parent task *p* after *a* has returned

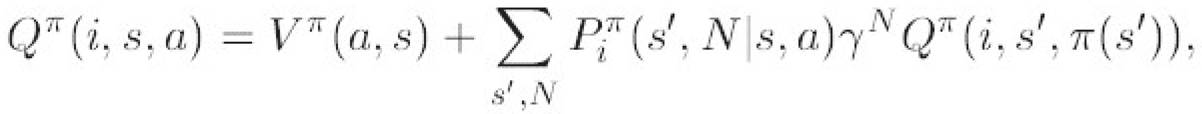
Hierarchical Structure

- MDP decomposed in task M0, … , Mn

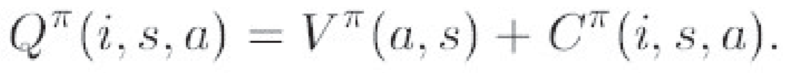


Theorem 1 *Given a task graph over tasks $M_0, \ldots, M_n$ and a hierarchical policy π, each subtask M_i defines a semi-Markov decision process with states S_i, actions A_i, probability transition function $P_i^\pi(s', N|s, a)$, and* expected *reward function $\overline{R}(s, a) = V^\pi(a, s)$, where $V^\pi(a, s)$ is the projected value function for child task M_a in state s. If a is a primitive action, $V^\pi(a, s)$ is defined as the expected immediate reward of executing a in s: $V^\pi(a, s) = \sum_{s'} P(s'|s, a)R(s'|s, a)$.*

- Q for the subtask i

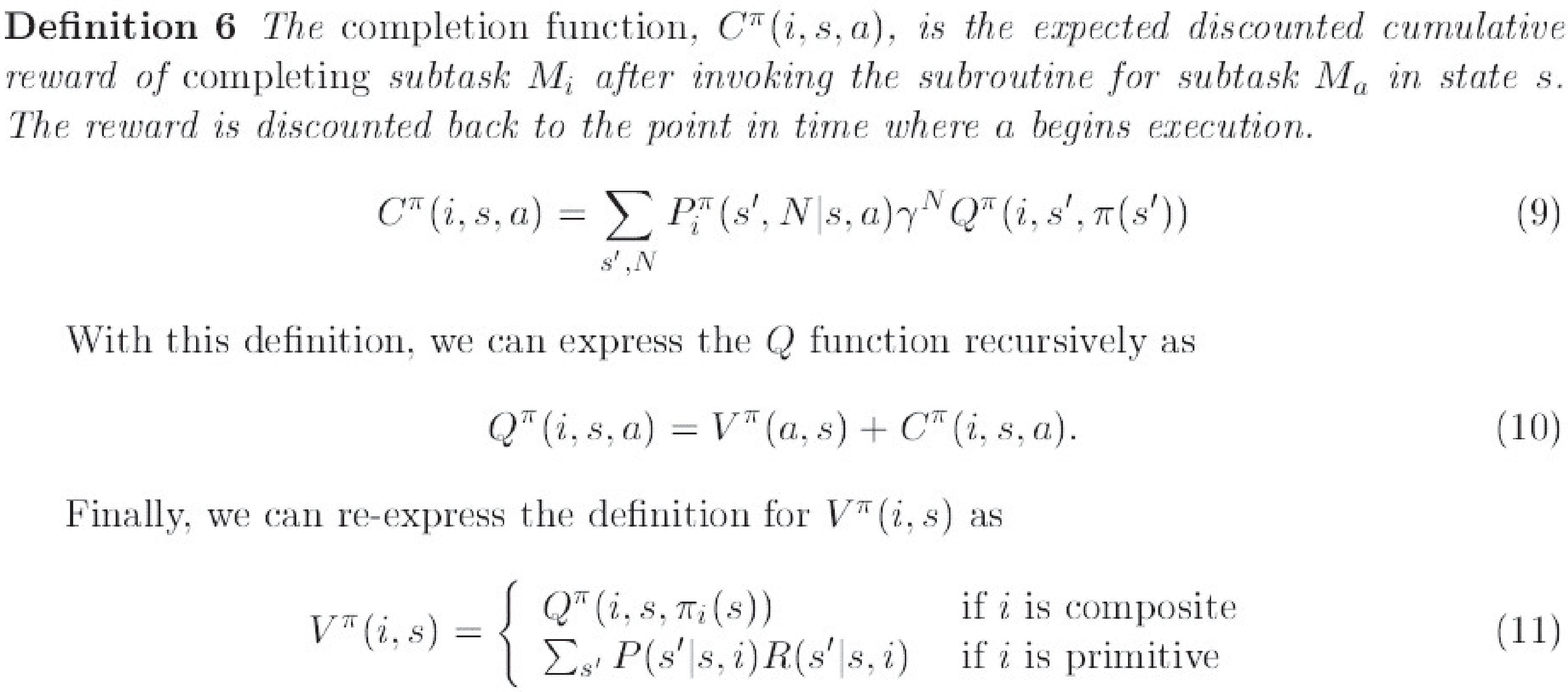


$$Q^\pi(i, s, a) = V^\pi(a, s) + \sum_{s', N} P_i^\pi(s', N|s, a)\gamma^N Q^\pi(i, s', \pi(s')),$$



$$Q^\pi(i, s, a) = V^\pi(a, s) + C^\pi(i, s, a).$$

Value Decomposition



Definition 6 *The* completion function, $C^\pi(i,s,a)$, *is the expected discounted cumulative reward of* completing *subtask* M_i *after invoking the subroutine for subtask* M_a *in state* s. *The reward is discounted back to the point in time where* a *begins execution.*

$$C^\pi(i,s,a) = \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi(s')) \tag{9}$$

With this definition, we can express the Q function recursively as

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a). \tag{10}$$

Finally, we can re-express the definition for $V^\pi(i,s)$ as

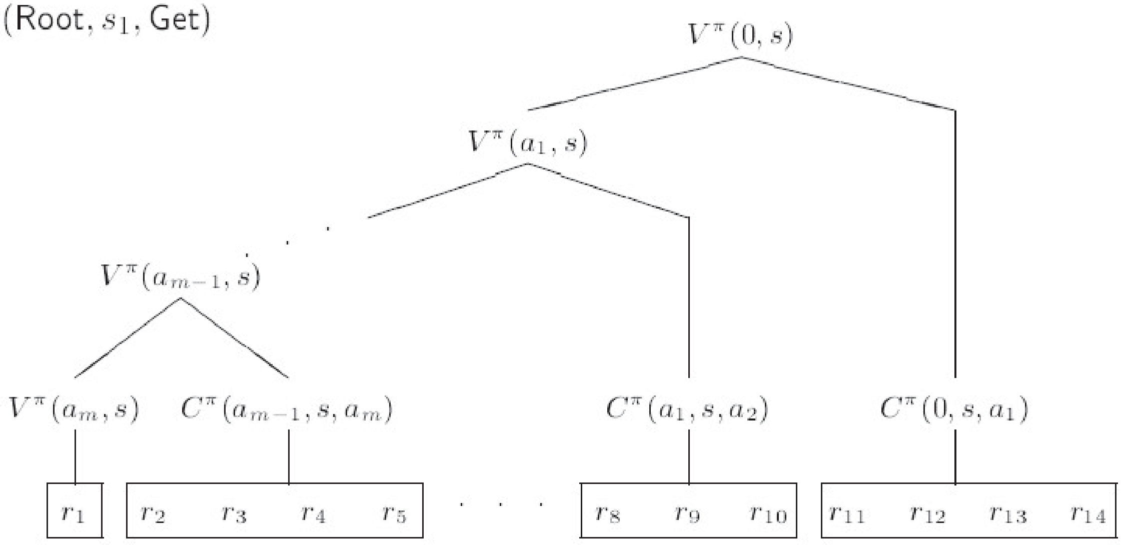
$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is primitive} \end{cases} \tag{11}$$

Value Decomposition

- The value function can be decomposed as follows

$$V^\pi(0,s) \quad = \quad V^\pi(a_m,s) + C^\pi(a_{m-1},s,a_m) + \ldots + C^\pi(a_1,s,a_2) + C^\pi(0,s,a_1)$$

$$\begin{aligned} V^\pi(\text{Root}, s_1) &= V^\pi(\text{North}, s_1) + C^\pi(\text{Navigate}(R), s_1, \text{North}) + \\ &\quad C^\pi(\text{Get}, s_1, \text{Navigate}(R)) + C^\pi(\text{Root}, s_1, \text{Get}) \\ &= -1 + 0 + -1 + 12 \\ &= 10 \end{aligned}$$



MAXQ Alg.

- An example

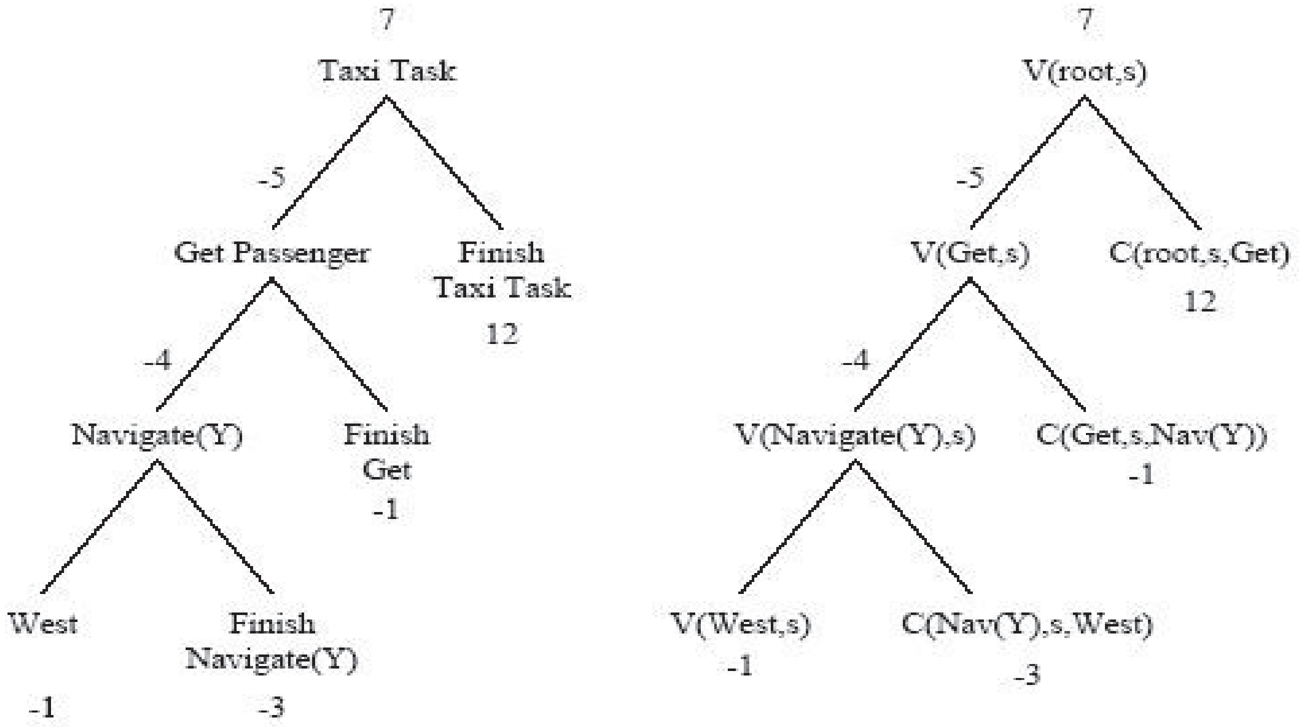
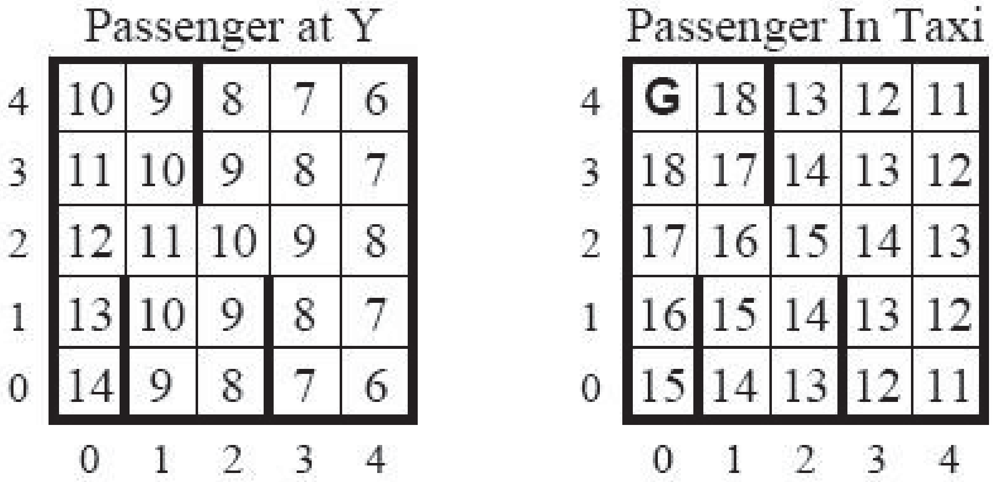


Fig. 5. An example of the MAXQ value function decomposition for the state in which the taxi is at location (2,2), the passenger is at (0,0), and wishes to get to (3,0). The left tree gives English descriptions, and the right tree uses formal notation.

MAXQ Alg.

$$\begin{aligned} V(\text{root}, s) = V(\text{west}, s) &+ C(\text{navigate}(Y), s, \text{west}) \\ &+ C(\text{get}, s, \text{navigate}(Y)) \\ &+ C(\text{root}, s, \text{get}). \end{aligned}$$



Passenger at Y

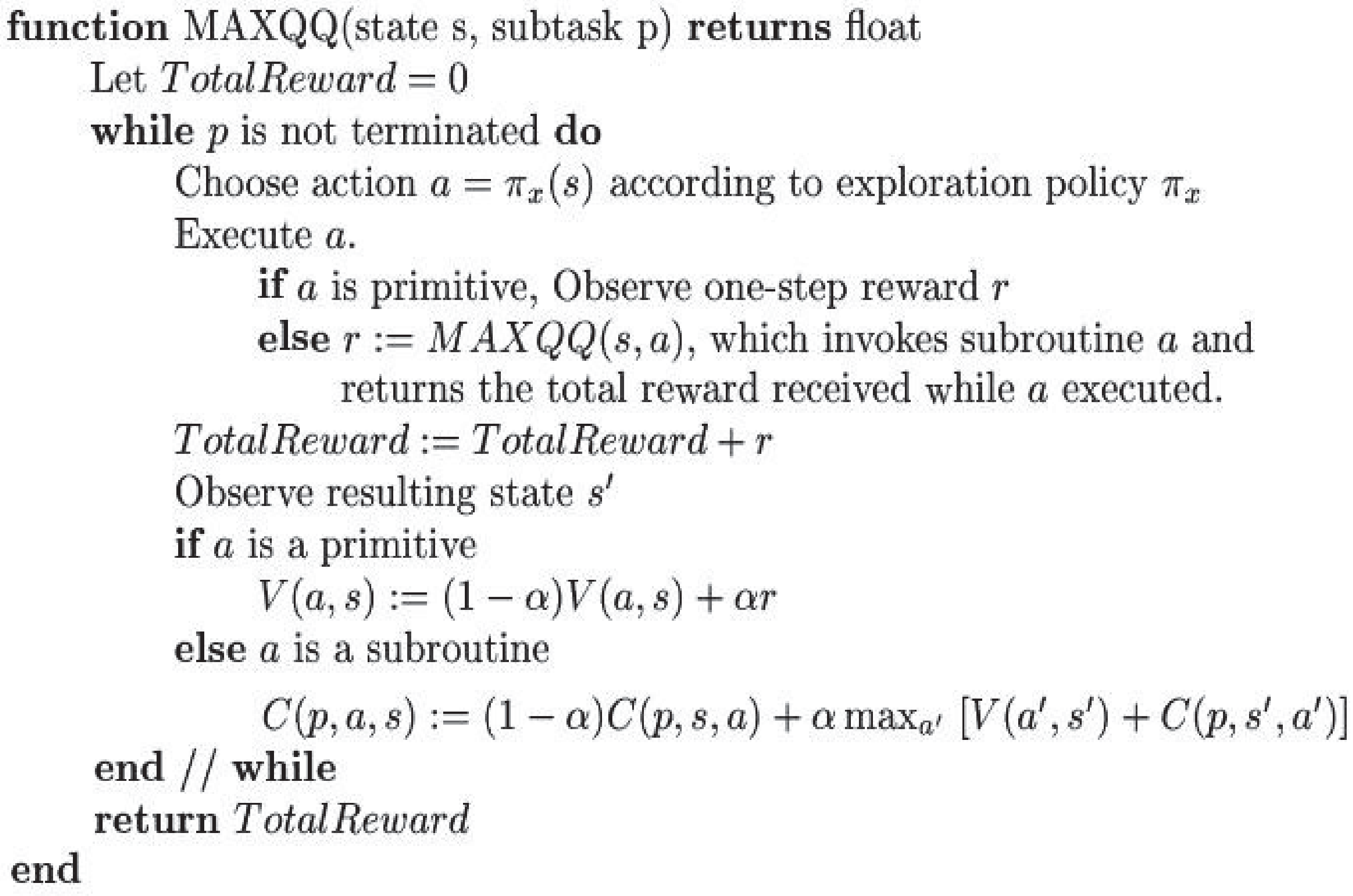
	0	1	2	3	4
4	10	9	8	7	6
3	11	10	9	8	7
2	12	11	10	9	8
1	13	10	9	8	7
0	14	9	8	7	6

Passenger In Taxi

	0	1	2	3	4
4	G	18	13	12	11
3	18	17	14	13	12
2	17	16	15	14	13
1	16	15	14	13	12
0	15	14	13	12	11

Fig. 4. Value function for the case where the passenger is at (0,0) (location Y) and wishes to get to (0,4) (location R).

MAXQQ Alg.



```
function MAXQQ(state s, subtask p) returns float
    Let $TotalReward = 0$
    while $p$ is not terminated do
        Choose action $a = \pi_x(s)$ according to exploration policy $\pi_x$
        Execute $a$.
            if $a$ is primitive, Observe one-step reward $r$
            else $r := MAXQQ(s, a)$, which invokes subroutine $a$ and
                returns the total reward received while $a$ executed.
        $TotalReward := TotalReward + r$
        Observe resulting state $s'$
        if $a$ is a primitive
            $V(a, s) := (1 - \alpha)V(a, s) + \alpha r$
        else $a$ is a subroutine
            $C(p, a, s) := (1 - \alpha)C(p, s, a) + \alpha \max_{a'} [V(a', s') + C(p, s', a')]$
    end // while
    return $TotalReward$
end
```

Optimality in HRL

Hierarchically optimal vs. recursively optimal

- Hierarchical optimality: The learnt policy is the best policy consistent with the given hierarchy. Task's policy depends not only on its children's policies, but also on its context.

- Recursive optimality: The policy for a parent task is optimal given the learnt policies of its children. (Context-free task's policy).

State Abstraction

Three fundamental forms

- Irrelevant variables

e.g. passenger location is irrelevant for the **navigate** and **put** subtasks and it thus could be ignored.

- Funnel abstraction

A funnel action is an action that causes a larger number of initial states to be mapped into a small number of resulting states. E.g., the ***navigate(t)*** action maps any state into a state where the taxi is at location *t*. This means the completion cost is independent of the location of the taxi—it is the same for all initial locations of the taxi.

State Abstraction

- Structure constraints

- E.g. if a task is terminated in a state s, then there is no need to represent its completion cost in that state
- Also, in some states, the termination predicate of the child task implies the termination predicate of the parent task

Effect

- reduce the amount memory to represent the Q-function.

14,000 q values required for flat Q-learning

3,000 for HSMQ (with the irrelevant-variable abstraction

632 for C() and V() in MAXQ

- learning faster

State Abstraction

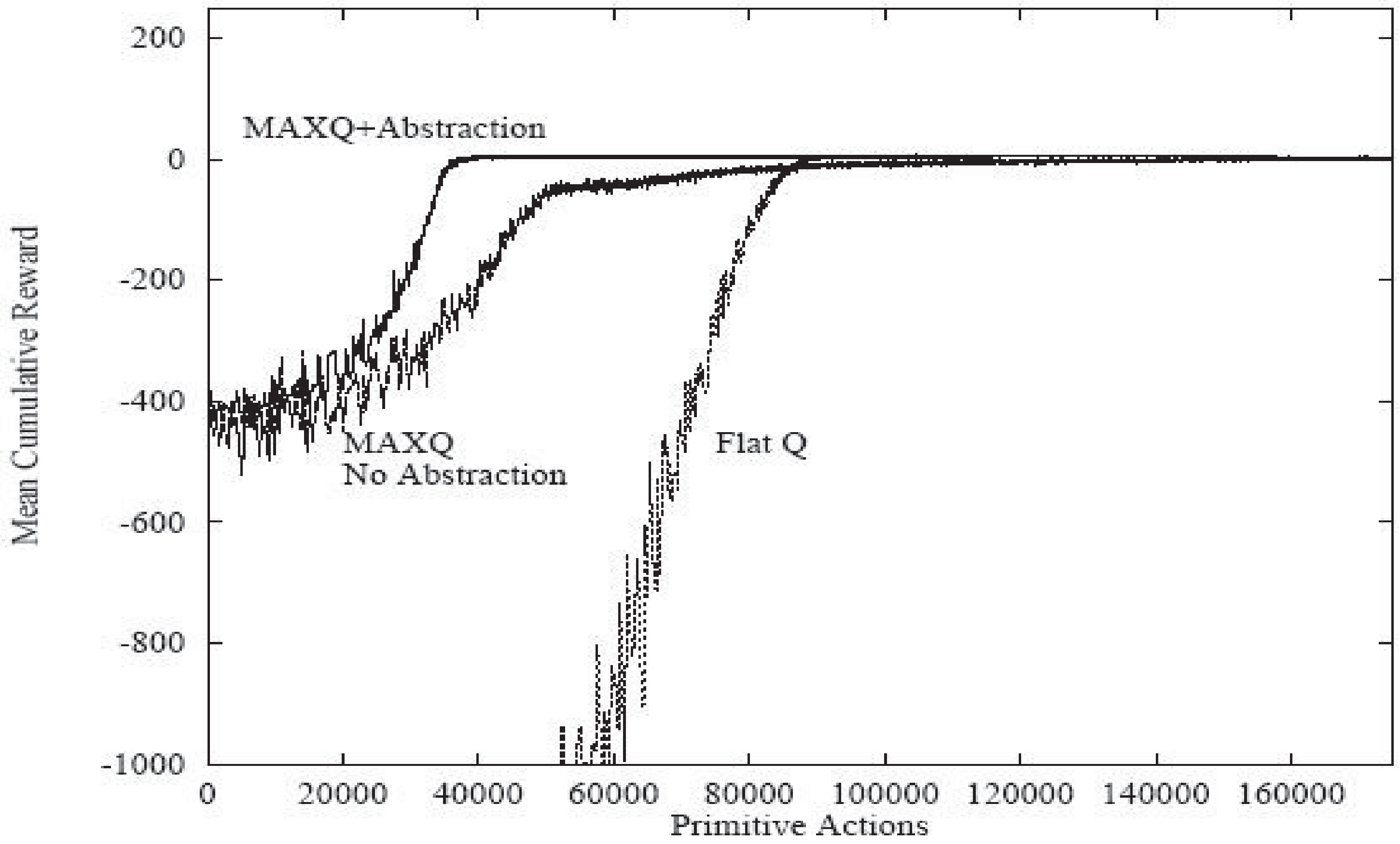


Fig. 7. Comparison of Flat Q learning, MAXQ Q learning with no state abstraction, and MAXQ Q learning with state abstraction on a noisy version of the taxi task.

Limitations

- Recursively optimal not necessarily optimal
- Model-free Q-learning

Model-based algorithms (that is, algorithms that try to learn *P(s'|s,a)* and *R(s'|s,a))* are generally much more efficient because they remember past experience rather than having to re-experience it.

References and Further Reading

- Sutton, R., Barto, A., *Reinforcement Learning: an Introduction*, The MIT Press

- Vlad Mnih, Koray Kavukcuoglu, et al. Human Level Control Through Deep Reinforcement Learning. Nature 2015

- Barto, A., Mahadevan, S., (2003) Recent Advances in Hierarchical Reinforcement Learning, *Discrete Event Dynamic Systems: Theory and Applications*, **13**(4):41-77