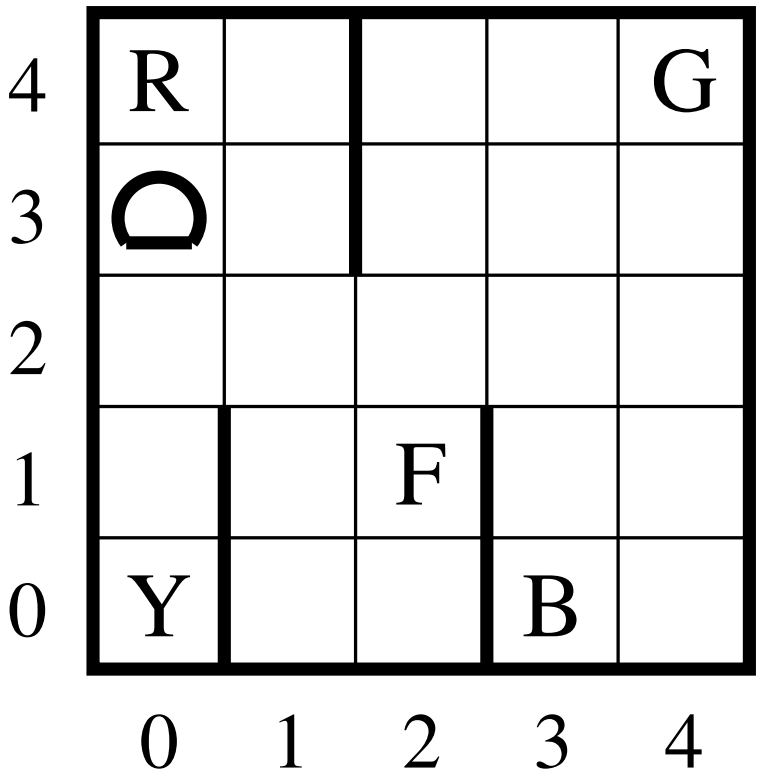
HIERARCHICAL REINFORCEMENT LEARNING

Denis Steckelmacher
dsteckel@vub.be

February 19, 2018

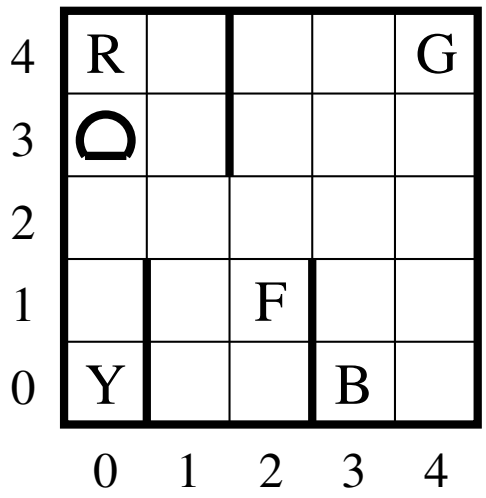
VRIJE UNIVERSITEIT BRUSSEL

THE PROBLEM



(Dietterich, 1998)

THE PROBLEM



(Dietterich, 1998)

7 actions (move, pickup, putdown, refill)

8750 states (cells, fuel, person)

THE PROBLEM

How to solve highly complex tasks?
(in a reasonable amount of time)

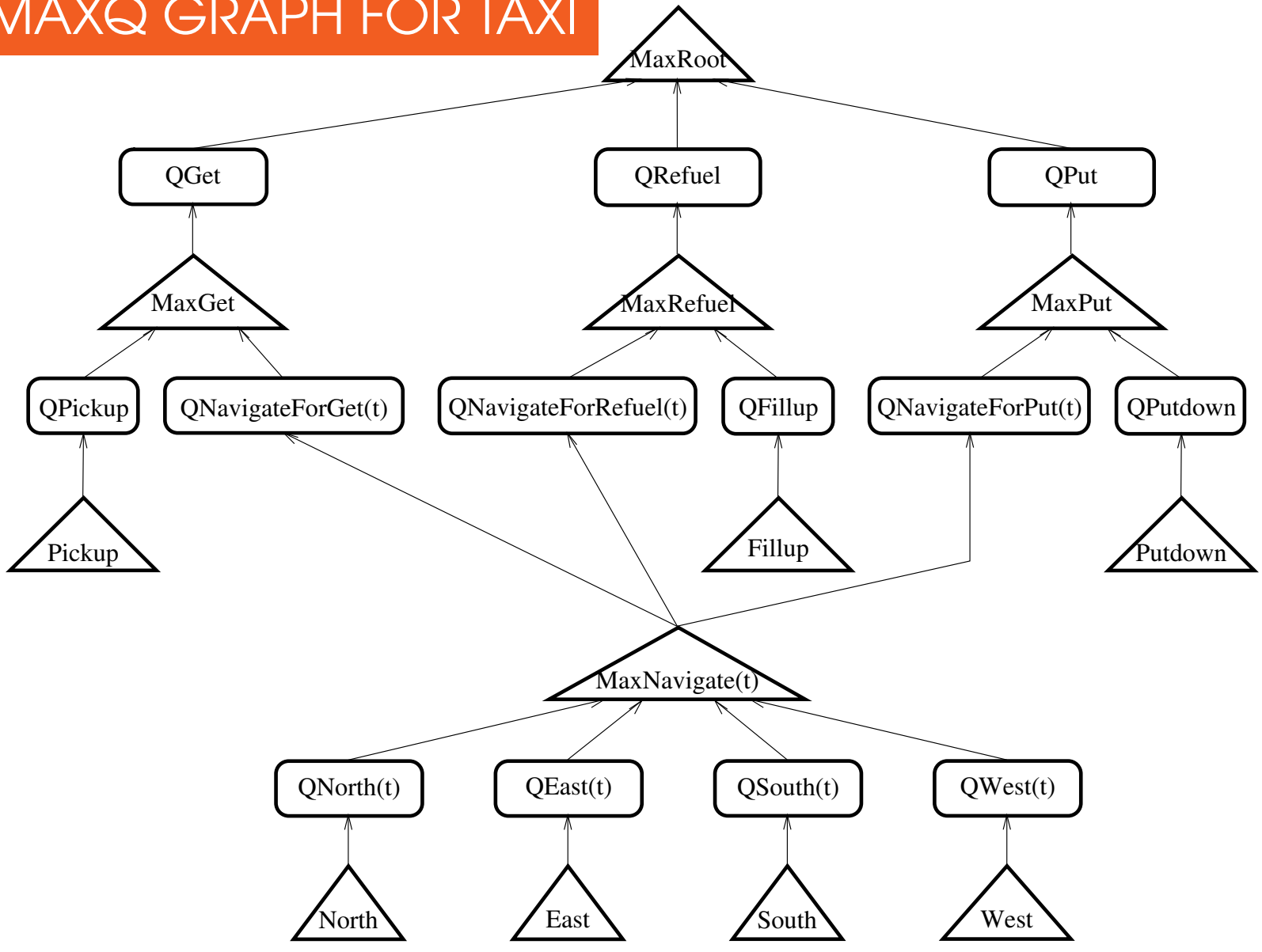
TWO SOLUTIONS

1. MAXQ
2. Options

MAXQ

1. Define a hierarchy of tasks and subtasks
2. Learn which subtask to activate in which situation

MAXQ GRAPH FOR TAXI



MAXQ (COMPLICATED) EQUATIONS

Max node a, $\langle \pi_a, S_a, T_a \rangle$, represents a policy π_a and the value of achieving a subtask.

Q node i, represents the value of performing i in the context of the whole task.

$$Q_i^\pi(s,a) = \overbrace{V_a^\pi(s)}^{\text{max node}} + \overbrace{C_i^\pi(s,a)}^{\text{Q node}}$$

$$V_a^\pi(s) = Q_a^\pi(s, \pi_a(s))$$

$$C_i^\pi(s,a) = \sum_{s'} P(s'|s,a) V_i^\pi(s')$$

LEARNING WITH MAXQ

Apply Q-Learning to V and C functions, then choose, at each time-step:

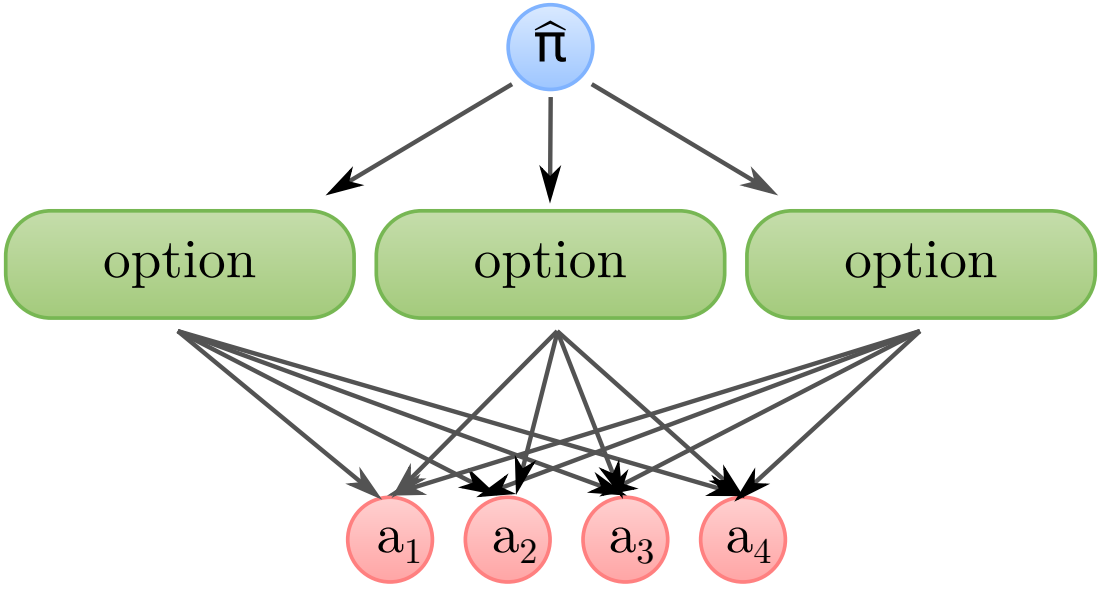
$$a^* = \text{argmax}_a(V_a(s') + C_i(s, a))$$

OPTIONS

A much simpler approach to hierarchical reinforcement learning!

1. Each option $\langle I_\omega, \pi_\omega, \beta_\omega \rangle$ is a self-contained policy for a subtask.
2. $\pi_\omega : S \to A$, the option policies
3. $\hat{\pi} : S \to O$, the top-level policy

OPTIONS



EXECUTING OPTIONS

1. Observe s_t, choose $\omega = \hat{\pi}(s_t)$
2. Execute $a_t = \pi_\omega(s_t)$
3. With probability $\beta_\omega(s_t)$, go back to 1.
4. With probability $1 - \beta_\omega(s_t)$, go back to 2.

LEARNING WITH OPTIONS

Learning with Options has been quite a challenge for the past 20 years.

1. Learn $\hat{\pi}$ with Q-Learning, the options are just "big“ actions. This is **inter-option learning**.
2. Learn π_ω with Q-Learning as well, learns to achieve a subgoal, this is **intra-option learning**.
3. Learn all policies at the same time using the **option-critic** architecture.

THE FLATTENED POLICY

$\hat{\pi}$, that selects options, and π_ω, that selects actions, can be merged together:

$$\pi(s, \omega) \to A + O$$

$$\pi(s, \emptyset) \to O \qquad \text{top-level, sets } \omega$$

$$\pi(s, \cdot) \to A \qquad \text{option policy, selects actions}$$

The option-critic directly learns $\pi(s, \omega)$.

THE FLATTENED POLICY WITH TERMINATION

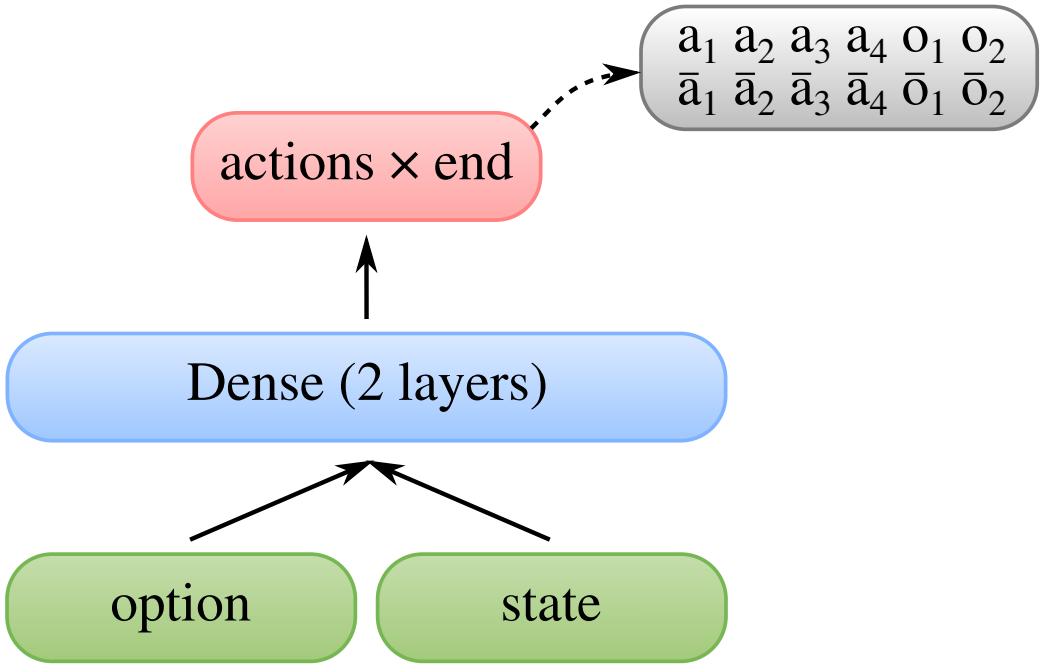
π selects actions or options, and can choose when an option terminates (β_ω):

$$\pi(s, \omega) \rightarrow (A + O) \times \{cont, term\}$$

Example: $\pi(\text{near door}, \text{go to door}) = \text{move forwards and terminate current option.}$

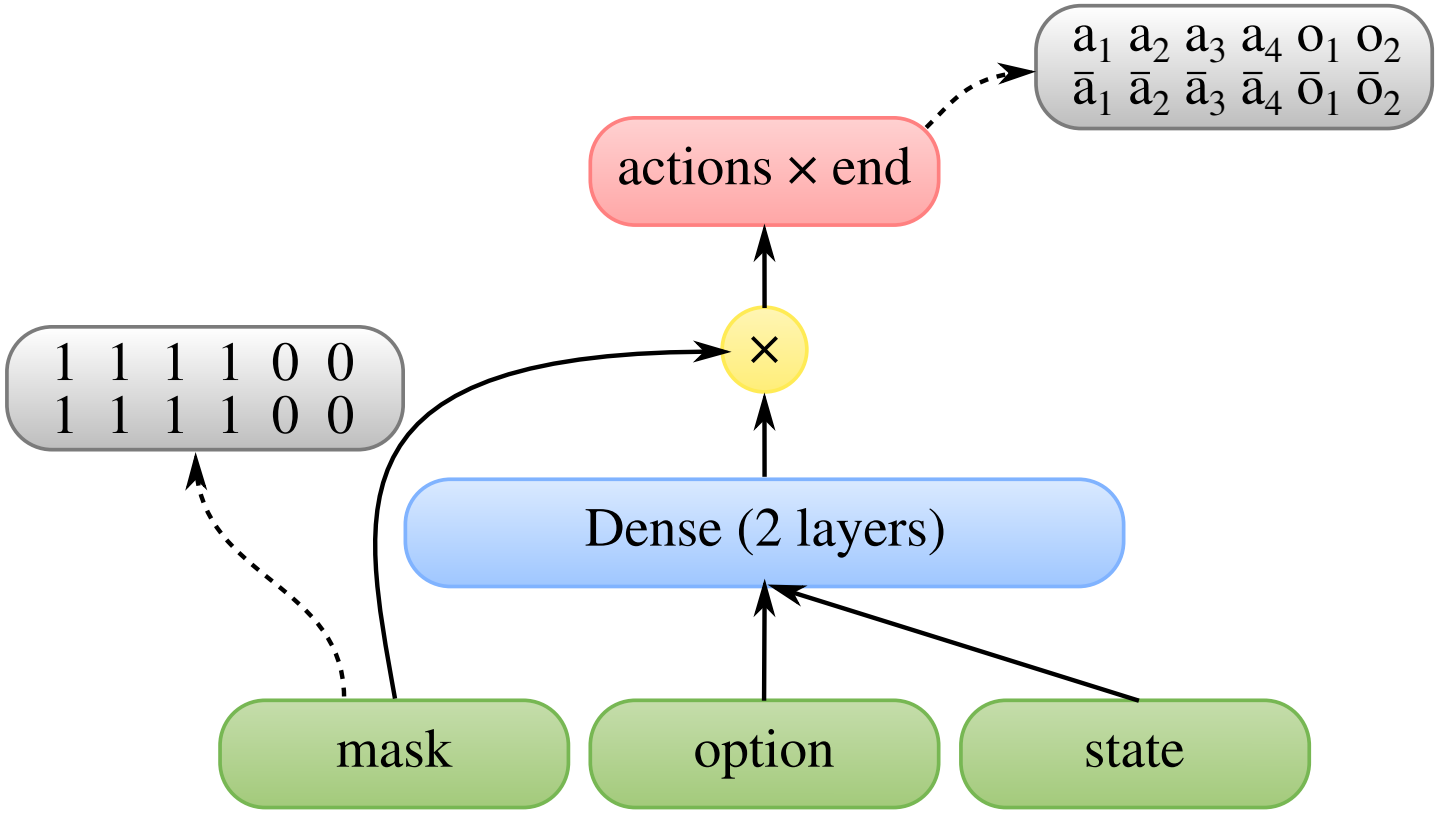
Example: $\pi(\text{in hallway}, \text{go to door}) = \text{move forwards and continue current option.}$

REPRESENTING THE FLATTENED POLICY



$$\pi(s, \omega) \rightarrow (A + O) \times \{cont, term\}$$

REPRESENTING THE FLATTENED POLICY



$$\pi(s, \omega, I_\omega) \rightarrow (A + O) \times \{cont, term\}$$

LEARNING THE FLATTENED POLICY

Use Policy Gradient to "magically“ learn the policy. Maximize:

$$L = \sum_t R_t \log \pi(a_t | s_t, \omega_t, l_{\omega_t})$$

$$R_t = \sum_{i=t}^{T} r_i$$

POLICY GRADIENT IN PRACTISE

```
state = keras.layers.Input((state_vars,))
ocurrent = keras.layers.Input((num_options,))
oallowed = keras.layers.Input((total_actions,))

stateoption = keras.layers.concatenate([state, ocurrent])
pi = make_function(stateoption, total_actions)
probas = make_probas(pi, oallowed)

model = keras.models.Model(
    inputs=[state, ocurrent, oallowed],
    outputs=[probas])
model.compile(optimizer=keras.optimizers.Adam())

pi_true = model.targets[0]
pi_pred = model.outputs[0]
logpi = K.log(pi_pred + epsilon)
grad = K.mean(pi_true * logpi)

model.total_loss = -grad
```

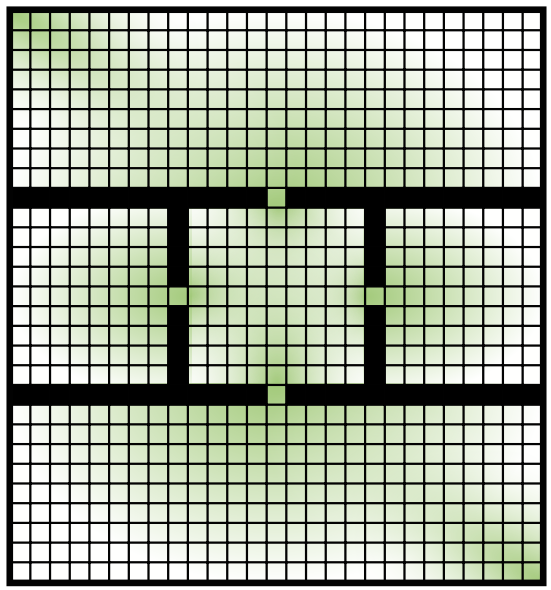
R_t

$\pi(s_t, a_t)$

LEARNING THE OPTIONS

Learning **the** options, not learning **with** options.

1. Discover bottleneck states and use them as **goal states**
2. Discover high changes in states and consider them as **salient states**
3. Factor big state-adjacency matrices (in discrete states)



USE OF OPTIONS: COMPLEX TASKS

https://www.youtube.com/watch?v=qxvk9DBboI0

A **complex behavior** on a robot is decomposed into **simpler subtasks**, encoded using options. The robot learns to combine the options.

USE OF OPTIONS: PARTIAL OBSERVABILITY

https://www.youtube.com/watch?v=VprJZEOD5NE

Options encode simple behaviors, in this case **going towards colored objects**. The agent learns a complex top-level policy, **using the options as memory**, that navigates towards different objects in order to solve a warehouse task: **bringing merchandise from terminals to a central carrier belt**.