Project : Reinforcement Learning (MVA)

Subgoal discovery in Reinforcement Learning

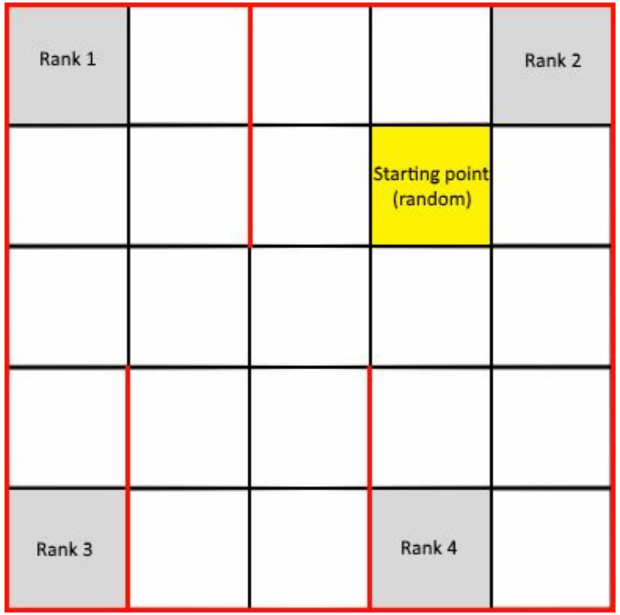
Yonatan DELORO and **Juliette RENGOT**

November, 2018 to January, 2019

Plan of the presentation

Problem definition : the taxi domain

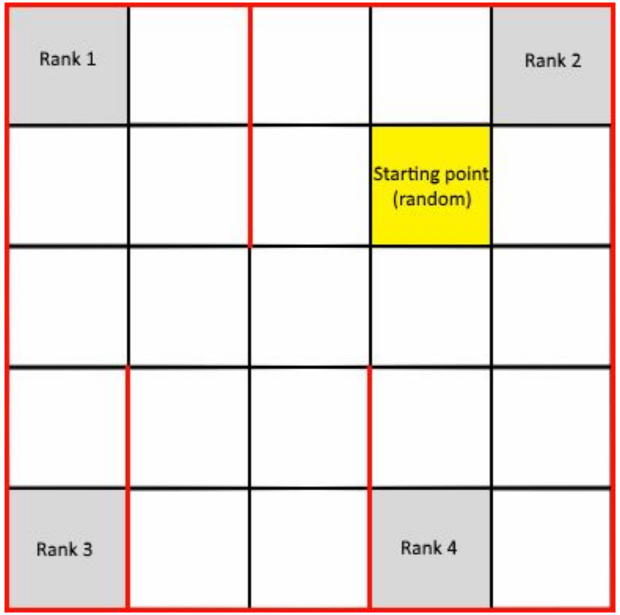
Original Version (Dietterich)



HEXQ test

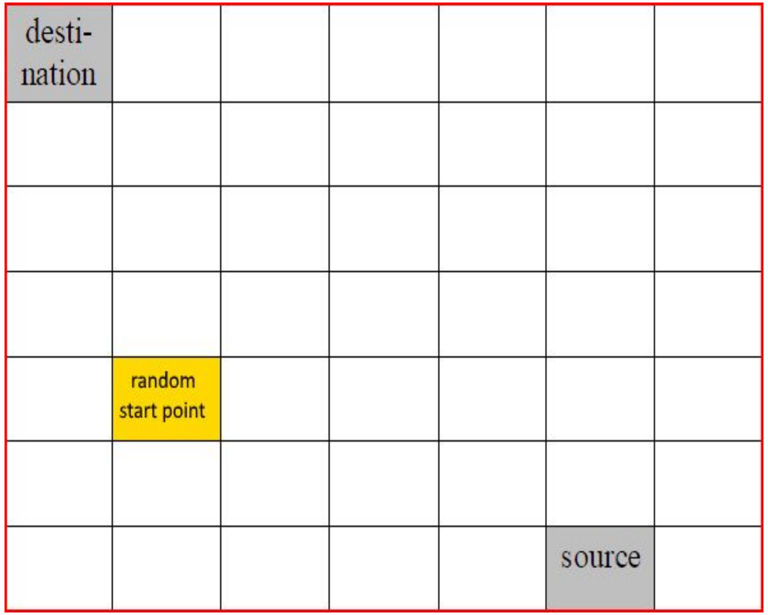
Problem definition : the taxi domain

Original Version (Dietterich)



HEXQ test

Simplified Version



DDMQ test

(“flat Q-learning”)

Standard Q-learning algorithm on a flat MDP formulation

The flat Q-learning solution

1° Formulating the Taxi Problem as a flat MDP

Space of states:

A state is defined by the triplet :

- the location of the cab in the grid
- the location of the passengers (in the cab / at which rank)
- the location of the destination (at which rank)

Space of primitive actions:

left, right, down, up pick up, put down

2° Applying standard Q-learning algorithm to solve this MDP

(“DDMQ”)

Macro Q-Learning with Diverse-Density based options

Amy McGovern and Andrew G. Barto. “Automatic Discovery Of Subgoals In Reinforcement Learning using Diverse Density”. In: Computer Science Department Faculty Publication Series. 8. (2001).

Theoretical aspects : *The big picture*

Ideas :

1) Mine the set of saved trajectories to :
   - define subgoal candidates as commonalities across successful saved trajectories (“bottlenecks”)
   - build policies aimed at reaching these subgoals

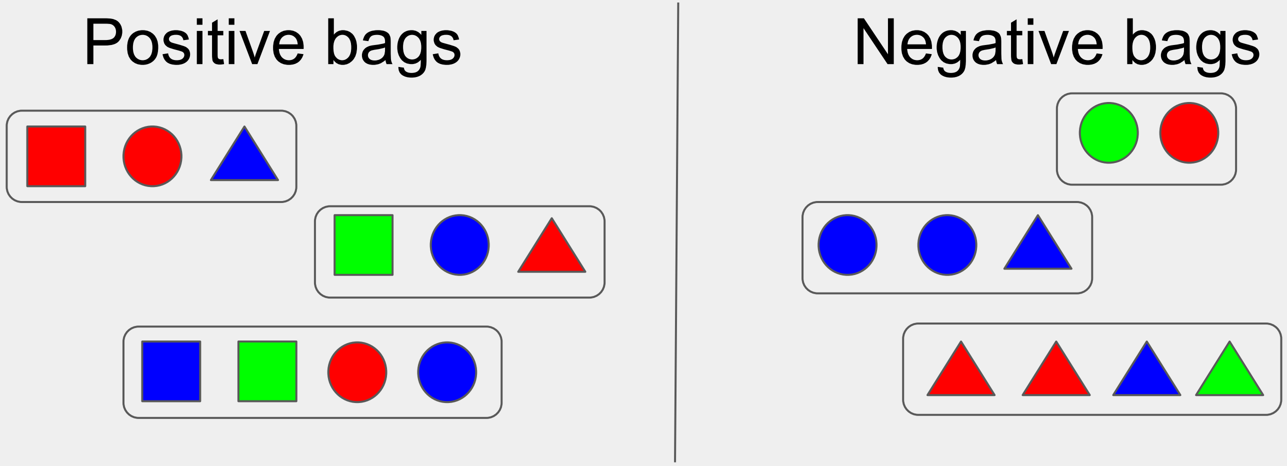
2) Enable the agent to follow and assess the built-in policies available from state, instead of primitive actions

Theoretical aspects :

A. How to discover subgoals ?

1. Looking for bottlenecks is a multiple instance learning problem

The MIL problem



Goal : find the target concept ☐

Theoretical aspects :

A. How to discover subgoals ?

1. Looking for bottlenecks is a multiple instance learning problem

-A trajectory is modelled as a bag of instances, where an instance typically corresponds to a state visited by the agent

-Successful trajectories = positive bags
-Unsuccessful trajectories = negative bags (with no positive instance by assumption)

-Target concepts = bottleneck region (type of instance responsible for the positivity of the bag it belongs to)

Theoretical aspects :

A. How to discover subgoals ?

<u>2. Solve the MIL problem using Diverse Density</u>

<u>Idea:</u> find the region with instances from the most positive bags and the least negative bags

ie. the concept with the highest diverse density as defined below

$$d(t) := Pr(t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-) = \prod_{i=1}^{n} Pr(t|B_i^+) \prod_{i=1}^{m} Pr(t|B_i^-)$$

$$Pr(t|B_i^-) = \prod_{1 \leq j \leq p} (1 - Pr(B_{ij}^- \in c_t))$$

$$Pr(t|B_i^+) = 1 - \prod_{1 \leq j \leq p} (1 - Pr(B_{ij}^+ \in c_t))$$

Theoretical aspects :

A. How to discover subgoals ?

<u>3. Keep only relevant diverse dense concepts</u>

1) Exclude concepts near terminal state of successful trajectories

2) Keep only DD peaks appearing in the first stages of learning and persisting throughout learning

At the end of a trajectory, the agent creates a new bag and looks for the DD peak. If concept c appears as peak, it updates the running average of how often c has appeared as a peak $m_c \leftarrow \lambda(m_c + 1)$

$$m_c > x\frac{\lambda}{1-\lambda} \Longrightarrow \text{c is selected as a sub-goal candidate}$$

Theoretical aspects :
B. Learning with discovered subgoal candidates

<u>1. Create an option for a subgoal candidate</u>

An option is a temporally-extended action which the agent executes until a termination condition is satisfied.

It is defined by:

- I : the option's input set of states
- π : the option's own policy
- β : the probability of termination in each state

Theoretical aspects : *B. Learning with discovered subgoal candidates*

<u>1. Create an option for a subgoal candidate</u>

We will use options to model built-in policies intended to reach subgoal candidates (target concepts)

In this purpose,

- I : contains the set of states visited on a trajectory during a time window before reaching the subgoal
- π : is computed based on a value function learnt using experience replay and a relevant option’s reward
- β : is 1 if the subgoal is reached or we left the input set

Theoretical aspects : *B. Learning with discovered subgoal candidates*

2. Use Macro Q-Learning to learn both with primitive actions and options

Primitive action played at t

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

Option initiated at t and terminated at t+n

$$Q(s_t, m_t) \leftarrow Q(s_t, m_t) + \alpha \left(\sum_{i=1}^{n} r_{t+i} \gamma^{i-1} + \gamma^n \max_{a'} Q(s_{t+n}, a') - Q(s_t, m_t) \right)$$

Implementation choices

Considered concepts and subgoal candidates

- A concept is defined as reaching an individual state (x, y, p)
- $Pr(B_{ij} \in c_t) = \exp -||B_{ij} - c_t||^2$
- DD values of concepts computed using positive bags only

- Concepts near the final goal terminal state excluded
- Maximum of one option created per run

Implementation choices

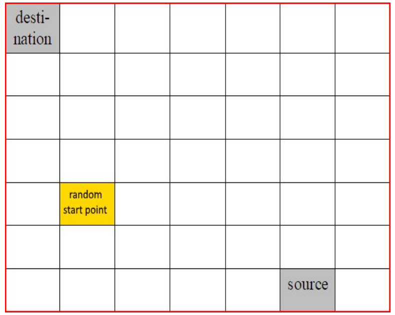
Considered metric for the probability to belong to a concept

1) First “agnostic” metric (used for report’s results) :

$$\boxed{||(x, y, p) - (x', y', p')||^2 = a_0(x - x')^2 + a_1(y - y')^2 + a_2(p - p')^2}$$

- Problem which still remains even with different scaling factors :

$$||(x, y, 1) - (x, y, 0)||^2 = cste$$



Implementation choices

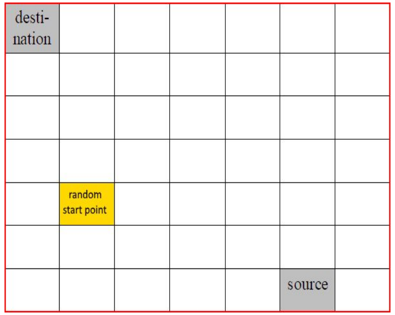
Considered metric for the probability to belong to a concept

1) First “agnostic” metric (used for report’s results) :

$$\boxed{||(x,y,p)-(x',y',p')||^2 = a_0(x-x')^2 + a_1(y-y')^2 + a_2(p-p')^2}$$

- Problem which still remains even with different scaling factors :

$$||(x,y,1)-(x,y,0)||^2 = cste$$

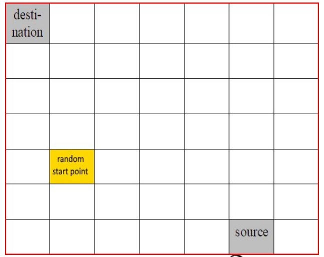


What we would prefer to find as distance instead :

$$||(x,y,0)-(x',y',0)||^2 = ||(x',y')-(x,y)||_2^2$$
$$||(x,y,1)-(x',y',1)||^2 = ||(x',y')-(x,y)||_2^2$$
$$||(x,y,0)-(x',y',1)||^2 = ||(x,y)-(x_{source},y_{source})||_2^2 + ||(x',y')-(x_{source},y_{source})||_2^2 + 1$$

Implementation choices

Considered metric for the probability to belong to a concept



What we would prefer to find as distance instead :

$$||(x,y,0)-(x',y',0)||^2 = ||(x',y')-(x,y)||_2^2$$

$$||(x,y,1)-(x',y',1)||^2 = ||(x',y')-(x,y)||_2^2$$

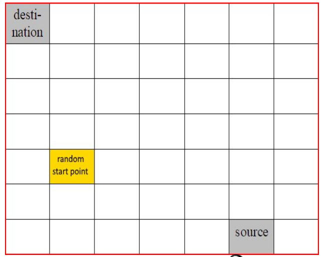
$$||(x,y,0)-(x',y',1)||^2 = ||(x,y)-(x_{source},y_{source})||_2^2 + ||(x',y')-(x_{source},y_{source})||_2^2 + 1$$

2) Second metric, changing the state representation (“task relevant”)

$$s = \begin{bmatrix} x \\ y \\ p \end{bmatrix} \Rightarrow \begin{bmatrix} p & \times & (x - x_{source}) \\ (1-p) & \times & (x - x_{source}) \\ p & \times & (y - y_{source}) \\ (1-p) & \times & (y - y_{source}) \end{bmatrix} := \phi(s) \qquad ||s - s'||^2 = ||\phi(s) - \phi(s')||_2^2$$

Implementation choices

Considered metric for the probability to belong to a concept



What we would prefer to find as distance instead :

$$||(x,y,0)-(x',y',0)||^2 = ||(x',y')-(x,y)||_2^2$$
$$||(x,y,1)-(x',y',1)||^2 = ||(x',y')-(x,y)||_2^2$$
$$||(x,y,0)-(x',y',1)||^2 = ||(x,y)-(x_{source},y_{source})||_2^2 + ||(x',y')-(x_{source},y_{source})||_2^2 + 1$$

2) Second metric, changing the state representation (“task relevant”)

$$s = \begin{bmatrix} x \\ y \\ p \end{bmatrix} \Rightarrow \begin{bmatrix} p & \times & (x-x_{source}) \\ (1-p) & \times & (x-x_{source}) \\ p & \times & (y-y_{source}) \\ (1-p) & \times & (y-y_{source}) \end{bmatrix} := \phi(s) \qquad ||s-s'||^2 = ||\phi(s)-\phi(s')||_2^2$$

We could retrieve these four features in :

$$P_2\left(\begin{bmatrix} (x-x_{source}) \\ (y-y_{source}) \\ p \\ 1-p \end{bmatrix}\right) := \phi(s)$$

$$\boxed{||s-s'||^2 = \sum_k a_k^2(\phi(s)_k - \phi(s')_k)^2}$$

Implementation choices

<u>Computing the option's policy</u>

- Option's reward : 10 is subgoal reached, -1 otherwise (-1 if we leave the input set or if we do not still reach the subgoal)

- Option's value function computed with Q-learning algorithm using option's reward and experience replay over parts of saved trajectories (time window before reaching subgoal)

Results *- experimental setup*

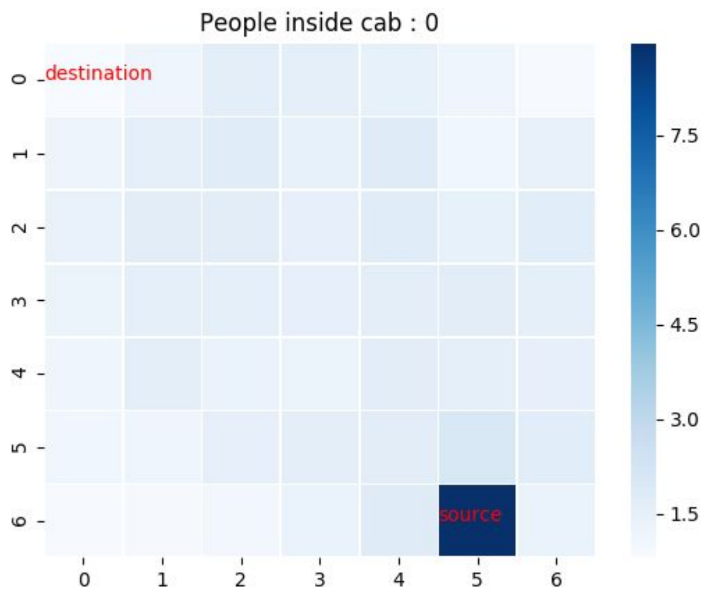
For both Q-learning and DDMQ :
- *Environment* : discount factor of 0.95
- *Trajectories* : N = 500, Tmax = 1000
- *Policy throughout learning* : eps-greedy policy
  eps_init =0.4, increase of 0.1 each 100 iters until 0.9
- *Learning rate* : adaptive
  (update of Q(s,a) : inverse of the number of visits of (s,a))

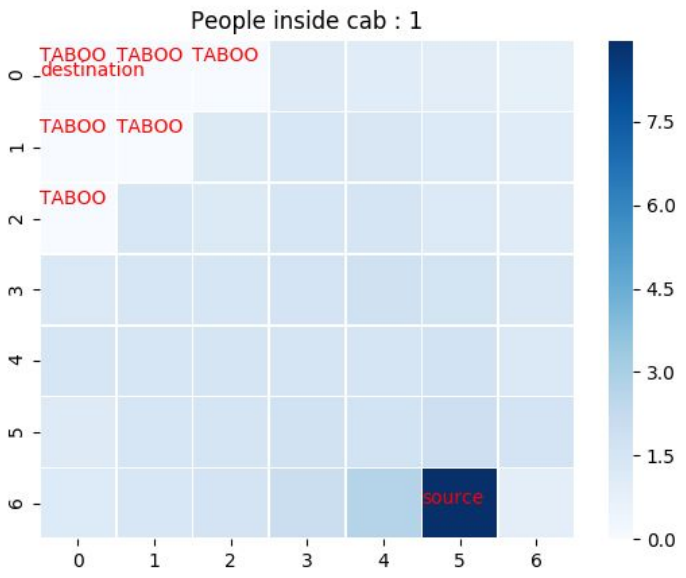
DDMQ hyperparameters :
- Time window of size 5 before reaching subgoal considered for computing option's input set and policy
- Discount factor for option reward = 0.95
- Peak concept frequency parameters : (λ, x) = (0.9 ; 0.9999)

$m_c \leftarrow \lambda(m_c + 1) \quad m_c > x\frac{\lambda}{1-\lambda}$

Results *- DDMQ*

"agnostic" metric

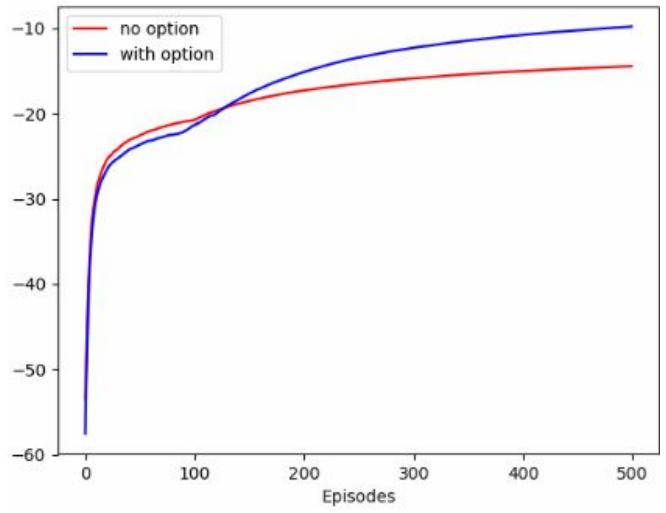




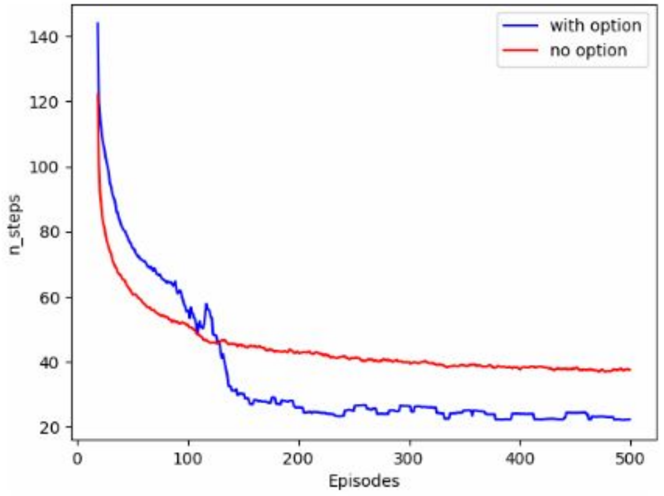
(a) Running average m_c of how often each concept c (taxi location, people presence in the cab) appeared as a peak at episode 50. Average over 30 runs of DDMQ

Results - *DDMQ vs. basic Q-learning*

"agnostic" metric



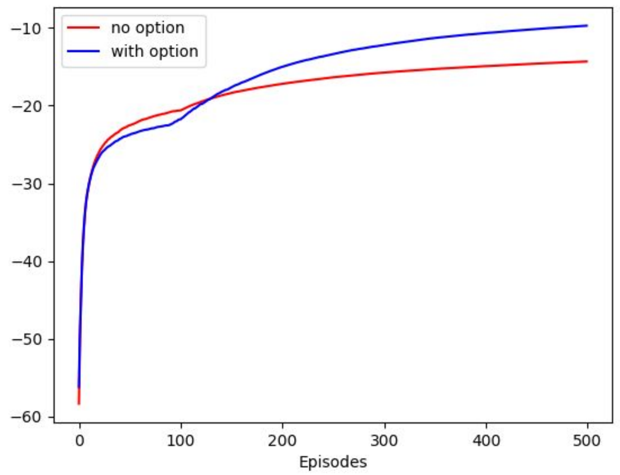
(b) For each algorithm, at a given episode, we plot the average of the discounted cumulative rewards across all previous episodes.



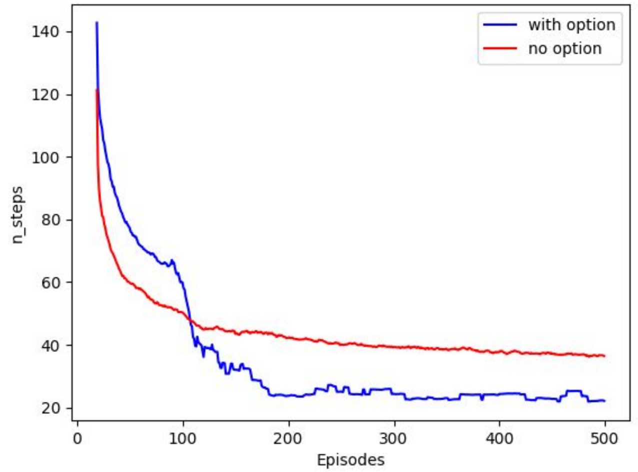
(c) Number of steps to reach the goal throughout the learning. Curbs are average over 30 runs for each of the two algorithms, and moving average over 20 episodes was then applied.

Results - *DDMQ vs. basic Q-learning*

"task-relevant" metric



(b) For each algorithm, at a given episode, we plot the average of the discounted cumulative rewards across all previous episodes.



(c) Number of steps to reach the goal throughout the learning. Curbs are average over 30 runs for each of the two algorithms, and moving average over 20 episodes was then applied.

HEXQ algorithm

Bernhard Hengst. “Discovering Hierarchy in Reinforcement Learning with HEXQ”. In: Proceedings of the Nineteenth International Conference on Machine Learning (July 2002), pp. 243–250.

Theoretical aspects :
Automatic hierarchical decomposition

<u>Idea</u> : Decompose a Markov Decision Process by dividing the state space into nested sub-MDP regions.

<u>Conditions</u> :

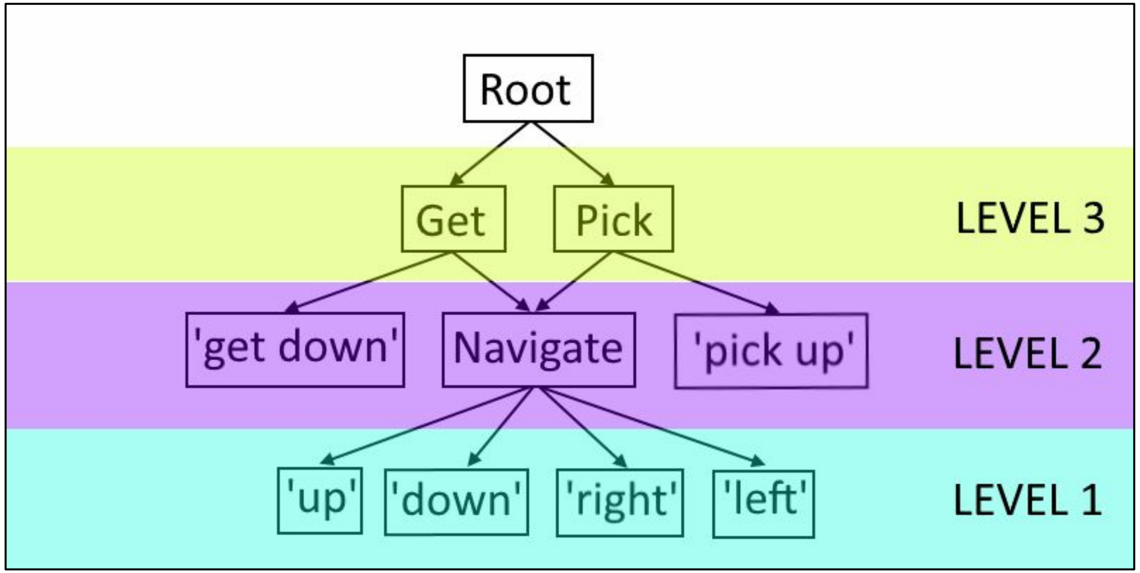
- ➔ variables change ± frequently
- ➔ transition properties retain
- ➔ interface between regions can be controlled

Theoretical aspects : *Automatic hierarchical decomposition*

Taxi problem

→ 3 state variables :

- destination rank $\in [\![0, 3]\!]$
- passenger source $\in [\![0, 4]\!]$
- taxi location $\in [\![0, 24]\!]$



Theoretical aspects :
State abstraction

$$\boxed{s^{e+1} = |r^e|\, x^j + r^e}$$

with s^{e+1} = abstract state value at level *e+1*

$|r^e|$ = number of regions at level e

r^e = region label from level e

x^j = next most frequent state variable value

Theoretical aspects : *Action abstraction*

The abstract actions available in each of these abstract states are the policies leading to region exits at the level below.

Taxi problem :

- 8 exits from bottom level

(s^1=0, a=pick up), (s^1=0, a=put down)
(s^1=4, a=pick up), (s^1=4, a=put down)
(s^1=20,a=pickup), (s^1=20,a=put down)
(s^1=23,a=pickup), (s^1=23,a=put down)

- 4 exits from next level :

(s^2=0, (s^1 = 0, a=put down))
(s^2=4, (s^1 = 0, a=put down))
(s^2=0, (s^1= 20, a=put down))
(s^2=0, (s^1= 23, a=put down))

Theoretical aspects :
The Value Function

$$\boxed{Q_{em}^{*}(s^{e}, a) = \sum_{s'} T_{se\ s'}^{a} (R_{se}^{a} + V_{em}^{*}(s'))}$$

with $T_{se\ s'}^{a} = \Pr(s'|s^{e}, a)$

R_{se}^{a} = E(primitive reward after $a|s^{e}$, a)

s' the hierarchical next state

$$\boxed{V^{*}_{em}(s) = \max_{a} (V^{*}_{e-1,m\_\{e-1\}(a)}(s) + Q^{*}_{em}(s^{e},a))}$$

with m_{e-1}(a) the sub-MDP implementing action a

Implementation choices
Hypothesis of simplification

<u>We suppose that the hierarchy is already constructed.</u>

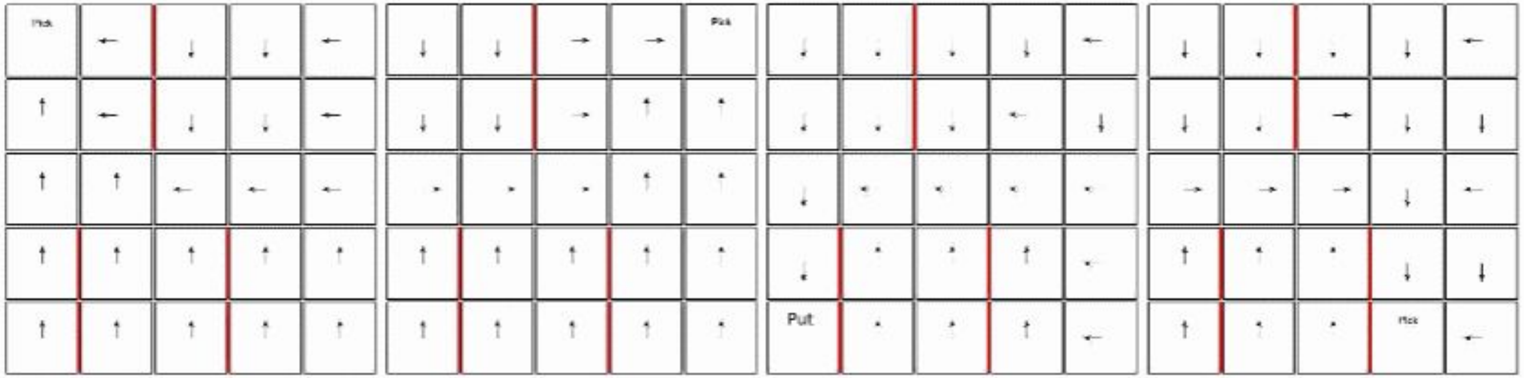
In theory, at the beginning of the training, HEXQ should order the variables and find exits at the first level before it can improve its performances over the taxi domain. In the article, this work takes 41 episodes.

Results **-** *experimental setup*

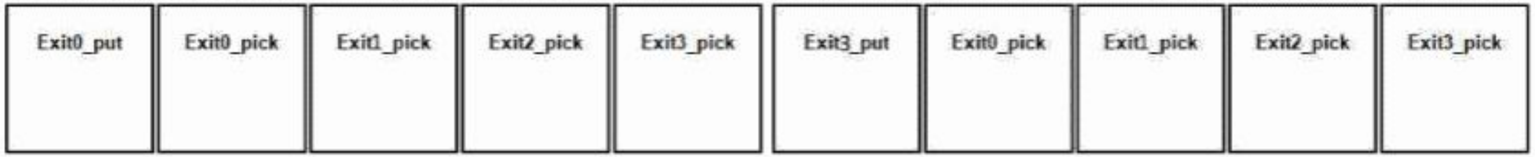
Parameters :

- Number of episode : N = 6000
- Time horizon : Tmax = 1000
- Exploration - exploitation ratio : $\varepsilon_{initial} = 0.1$
  → It increases of 0.1 each 200 episodes until 0.9
- *Learning rate* : adaptive
  → It is based on the inverse of the number of visits of (state, action)

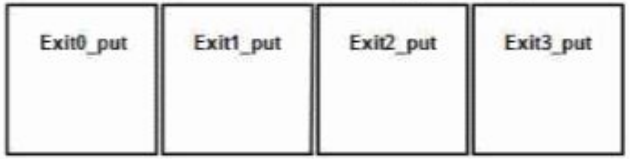
Results *- HEXQ policies*



(a) Policy at level 0 when the objective is 0 and people are not in the taxi. (b) Policy at level 0 when the objective is 4 and people are not in the taxi. (c) Policy at level 0 when the objective is 20 and people are in the taxi. (d) Policy at level 0 when the objective is 23 and people are not in the taxi.



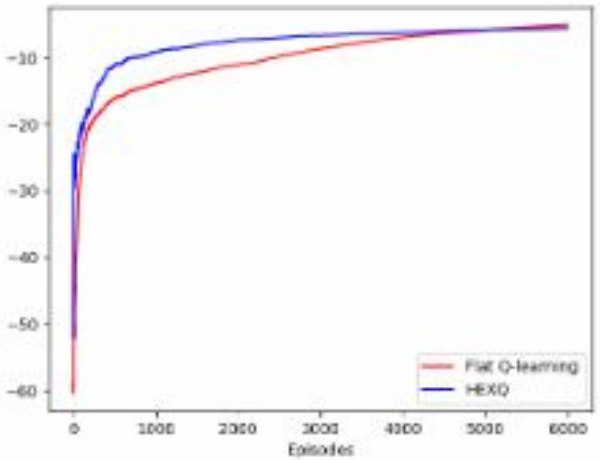
(e) Policy at level 1 when destination is 0. (f) Policy at level 1 when destination is 23.

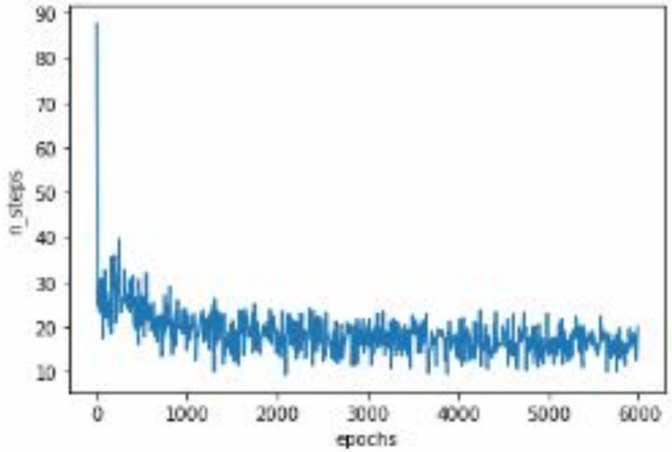


(g) Policy at level 2.

Results

HEXQ vs. flat Q-learning

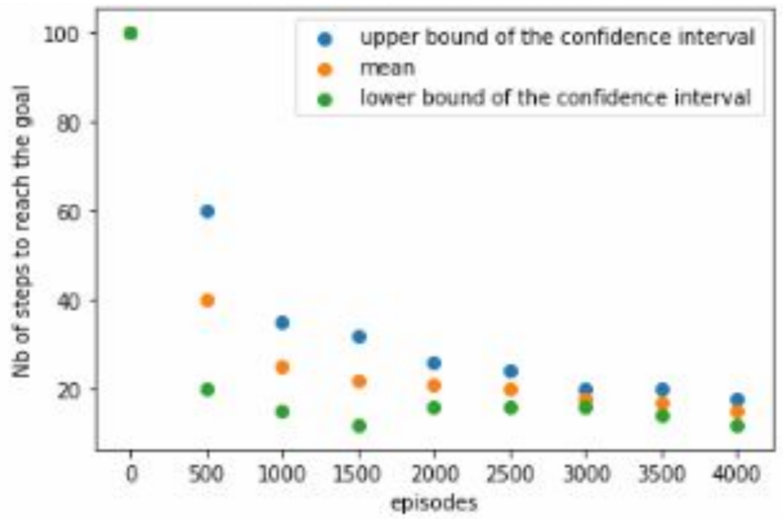


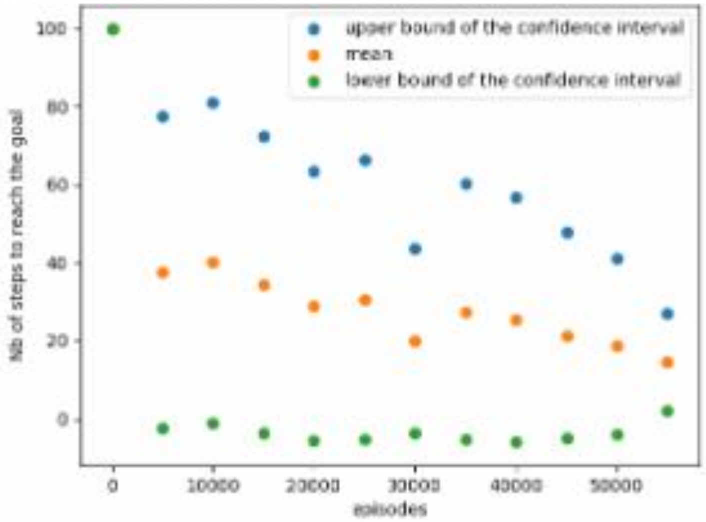


(a) For each algorithm, at a given episode, we plot the average of the discounted cumulative rewards across all previous episodes.

(b) Number of steps to reach the goal throughout the learning, for one run of HEXQ algorithm (moving average over 10 episodes was applied).

Results
HEXQ vs. flat Q-learning

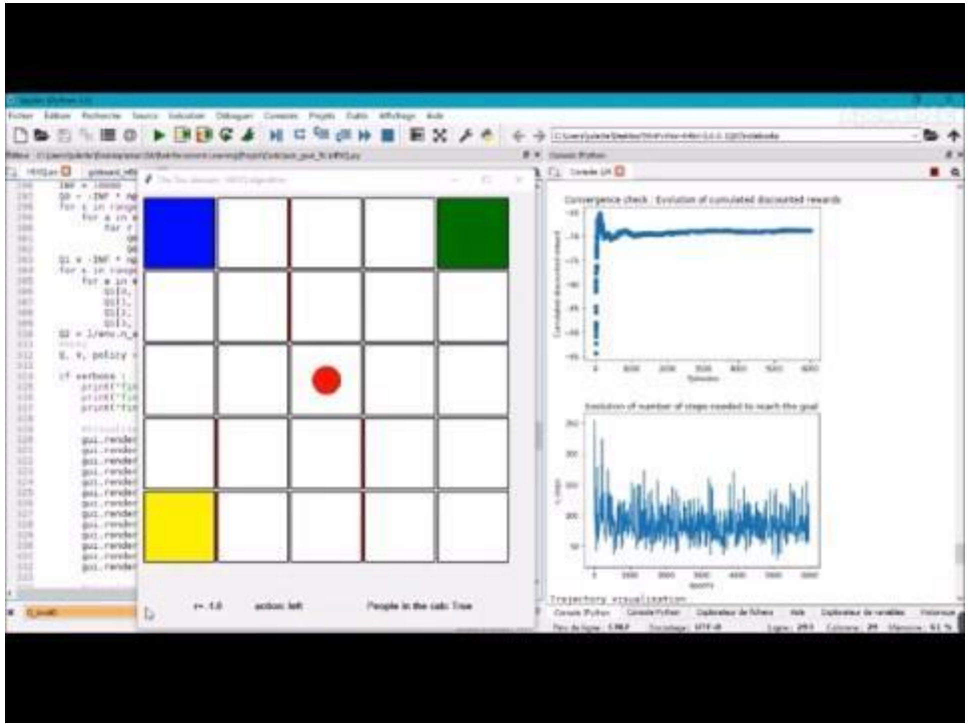




(c) HEXQ: Statistics for 100 trajectories, exploiting the learnt policy, of the number of steps to reach the goal. 68% confidence interval (+/- 1 std).

(d) Flat Q-learning: Statistics for 100 trajectories, exploiting the learnt policy, of the number of steps to reach the goal. 68% confidence interval (+/- 1 std).

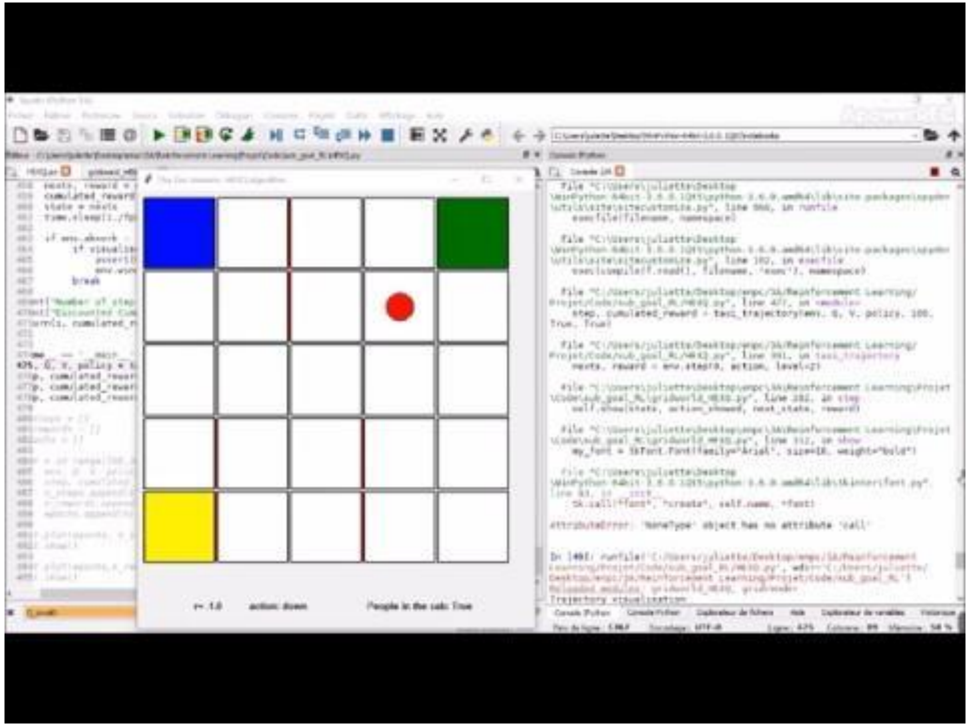
Results - HEXQ


Convergence check : Evolution of cumulated discounted rewards
Evolution of number of steps needed to reach the goal
r= -1.0
action: left
People in the cab: True

<u>Demo 1</u>:
Starting point = 20
passenger source = 4
destination rank = 0
Proba_success = 1

**We reach the goal
in 17 steps
with a reward of -12.61.**

Results - HEXQ



```
  File "C:\Users\juliette\Desktop
[illegible]
    execfile(filename, namespace)

  File "C:\Users\juliette\Desktop
[illegible]
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/juliette/Desktop/[illegible]/Reinforcement Learning/
Projet/Code/sub_goal_RL/HEXQ.py", line [illegible], in <module>
    step, cumulated_reward = taxi_trajectory(env, Q, V, policy, 100,
True, True)

  File "C:/Users/juliette/Desktop/[illegible]/Reinforcement Learning/
Projet/Code/sub_goal_RL/HEXQ.py", line [illegible], in taxi_trajectory
    nexts, reward = env.step(0, action, level=2)

  File "C:\Users\juliette\Desktop\[illegible]\Reinforcement Learning\Projet
\Code\sub_goal_RL\gridworld_HEXQ.py", line [illegible], in step
    self.show(state, action_showed, next_state, reward)

  File "C:\Users\juliette\Desktop\[illegible]\Reinforcement Learning\Projet
\Code\sub_goal_RL\gridworld_HEXQ.py", line [illegible], in show
    my_font = tkfont.Font(family='Arial', size=18, weight="bold")

  File "C:\Users\juliette\Desktop
[illegible]\tkinter\font.py",
line [illegible], in __init__
    tk.call("font", "create", self.name, *font)

AttributeError: 'NoneType' object has no attribute 'call'

In [illegible]: runfile('C:/Users/juliette/Desktop/[illegible]/Reinforcement
Learning/Projet/Code/sub_goal_RL/HEXQ.py', wdir='C:/Users/juliette/
Desktop/[illegible]/Reinforcement Learning/Projet/Code/sub_goal_RL')
Reloaded modules: gridworld_HEXQ, gridrender
Trajectory visualization
```

<u>Demo 2</u>:

Proba_success = 0.6

We reach the goal in 26 steps with a reward of -34.53.

Conclusion and discussion

<u>DDMQ</u>

Interesting when clear bottlenecks in the observation space

Limits :

1° assumes we can reach the main goal using only primitive actions first

2° challenging to define a relevant metric between instances

3° memory cost + number of hyperparameters

<u>HEXQ</u>

Interesting when repetition of sub-structures in the task

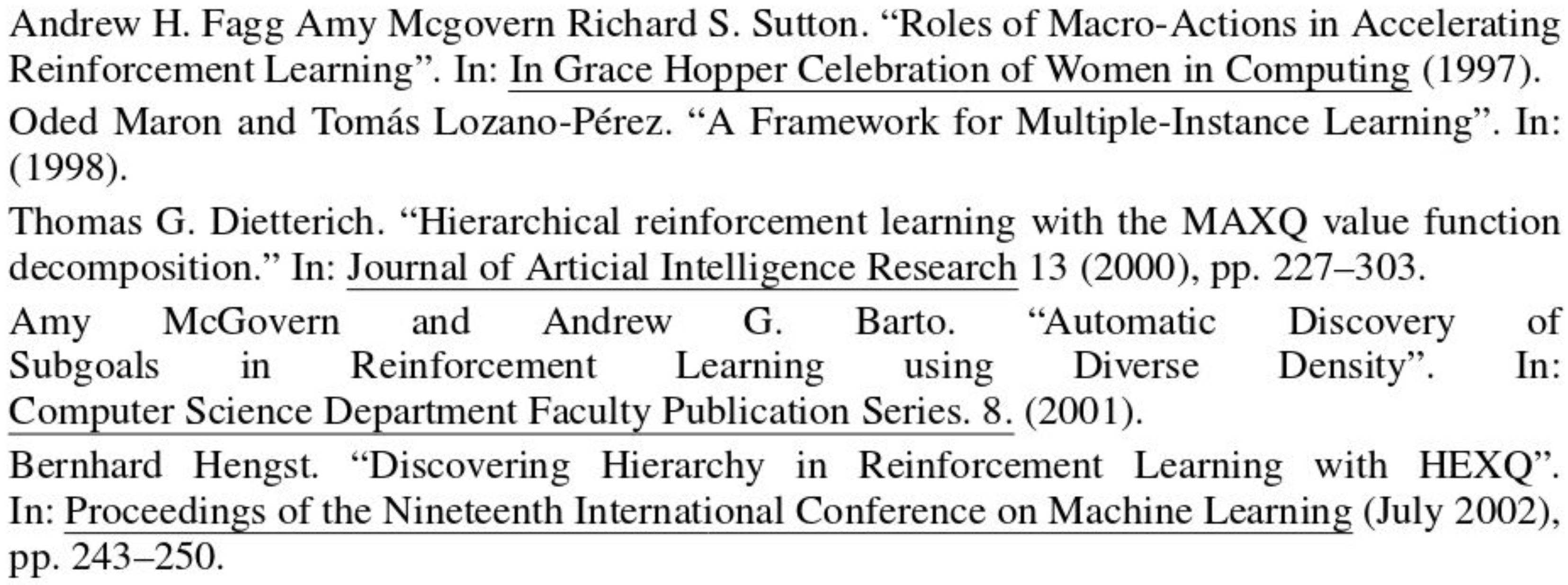
Limits :

1° assumes we can define some variable that vary at longer timescale than others

2° preliminary cost to build the hierarchy (variables frequency analysis and exits discovery)

Thank you for your attention
and for the proposal of this project !

References



Andrew H. Fagg Amy Mcgovern Richard S. Sutton. “Roles of Macro-Actions in Accelerating Reinforcement Learning”. In: In Grace Hopper Celebration of Women in Computing (1997).

Oded Maron and Tomás Lozano-Pérez. “A Framework for Multiple-Instance Learning”. In: (1998).

Thomas G. Dietterich. “Hierarchical reinforcement learning with the MAXQ value function decomposition.” In: Journal of Articial Intelligence Research 13 (2000), pp. 227–303.

Amy McGovern and Andrew G. Barto. “Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density”. In: Computer Science Department Faculty Publication Series. 8. (2001).

Bernhard Hengst. “Discovering Hierarchy in Reinforcement Learning with HEXQ”. In: Proceedings of the Nineteenth International Conference on Machine Learning (July 2002), pp. 243–250.

What we have learnt

There are many ways we can formulate the same RL task, with very variable impacts on the performances

- We should think carefully to the type of actions we define :

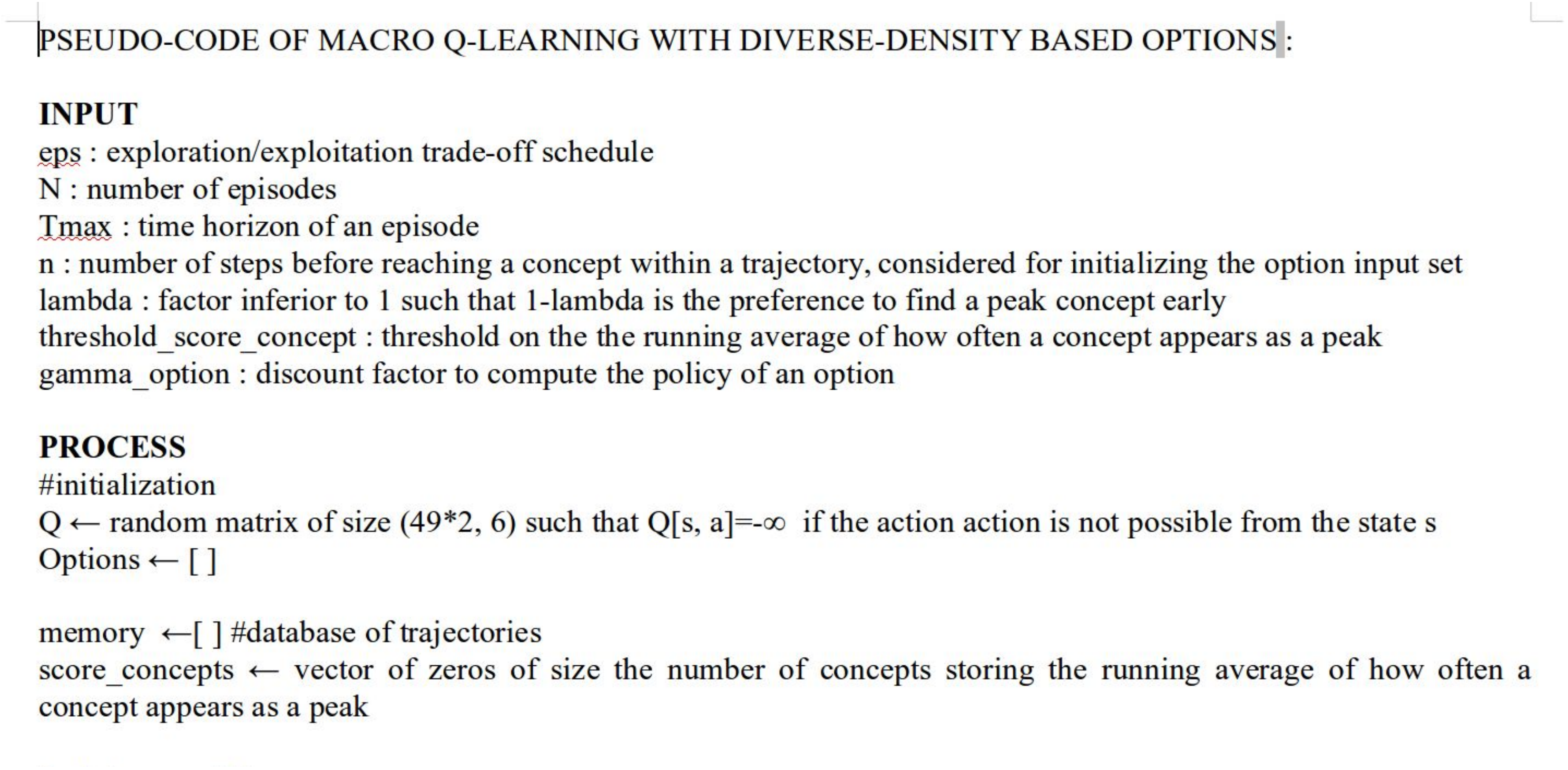
-> only primitive actions (flat Q-learning)

-> extended in time (DDMQ)

-> at different abstraction levels (HEXQ)

- How we represent a state can be also important (cf. choice of metric in DDMQ)

Pseudo-code of DDMQ



PSEUDO-CODE OF MACRO Q-LEARNING WITH DIVERSE-DENSITY BASED OPTIONS :

INPUT

eps : exploration/exploitation trade-off schedule
N : number of episodes
Tmax : time horizon of an episode
n : number of steps before reaching a concept within a trajectory, considered for initializing the option input set
lambda : factor inferior to 1 such that 1-lambda is the preference to find a peak concept early
threshold_score_concept : threshold on the the running average of how often a concept appears as a peak
gamma_option : discount factor to compute the policy of an option

PROCESS

```
#initialization
Q ← random matrix of size (49*2, 6) such that Q[s, a]=-∞  if the action action is not possible from the state s
Options ← [ ]

memory  ←[ ] #database of trajectories
score_concepts ← vector of zeros of size the number of concepts storing the running average of how often a
concept appears as a peak
```

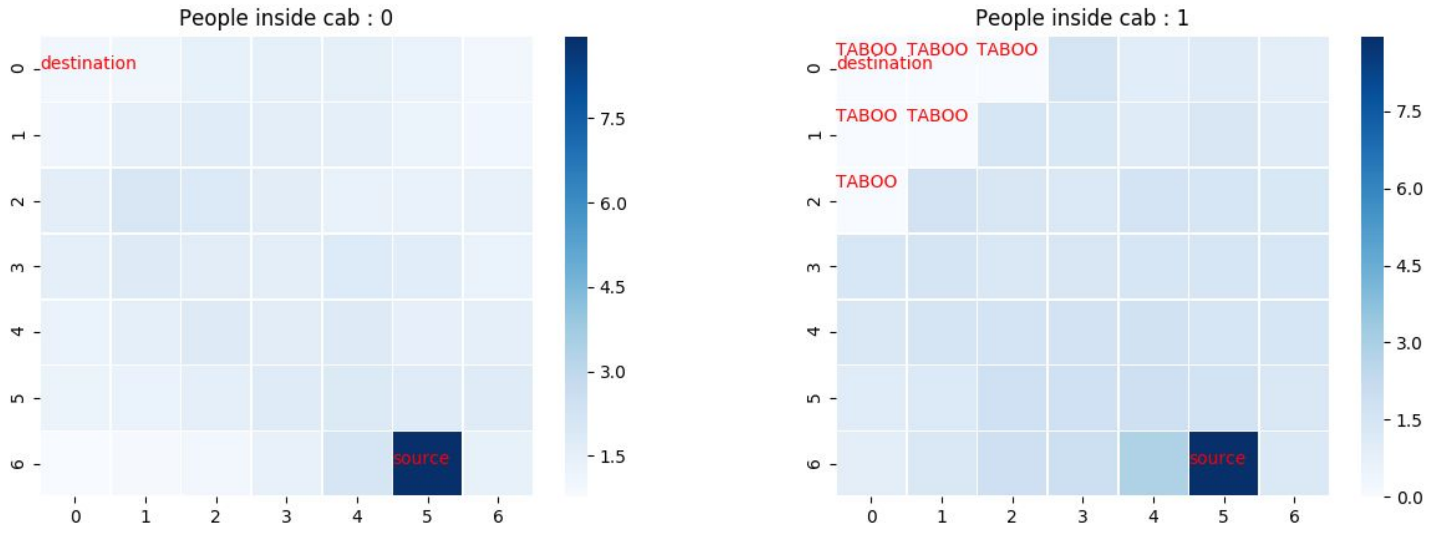
```
for k in range(N):
        state ← new initial state
        trajectory ← [ ]
        #MACRO Q-LEARNING
        while (t<Tmax) and (env is not solved):
                action ← choose an action (primitive/option), randomly with a probability eps, otherwise the one
                play action and update Q[state_0, action]
        add trajectory to memory

        #DIVERSE-DENSITY BASED SUBGOAL DISCOVERY TO CREATE NEW OPTION
        create positive bag from trajectory if successful
        compute the diverse density from all the saved positive bags, and search for diverse density peaks
        for each peak concept c found:
                score_concepts[c] ← lambda * (score_concepts[c]+1)
                if (score_concepts[c] > threshold_score_concept) and (c not in taboo states):
                        mark concept as subgoal candidate
        select at random one concept c among the subgoal candidates #max one option created per episode
        create a new option o = <I, \pi, \beta> of reaching concept c, and add it to Options
        add a column Qc to the Q matrix for the newly created option,  such that  such that Qc[s] =-∞  if the state s
        is not in the option input set

pi ← compute policy from Q matrix
```

OUTPUT

```
    final policy : pi #associate the index of the primitive action or option to play from each possible state
    list of options : Options: #definitions of options (input set, option policy, terminal condition)
```

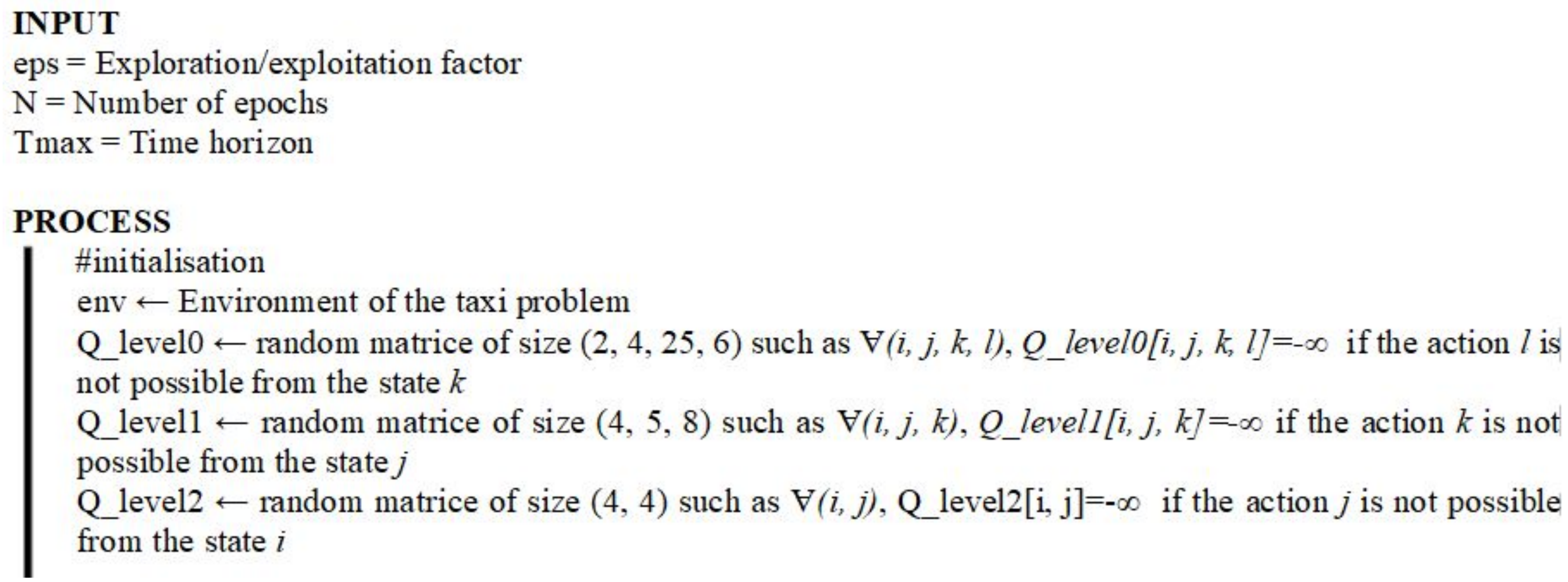
***Results** - DDMQ*

"task-relevant" metric



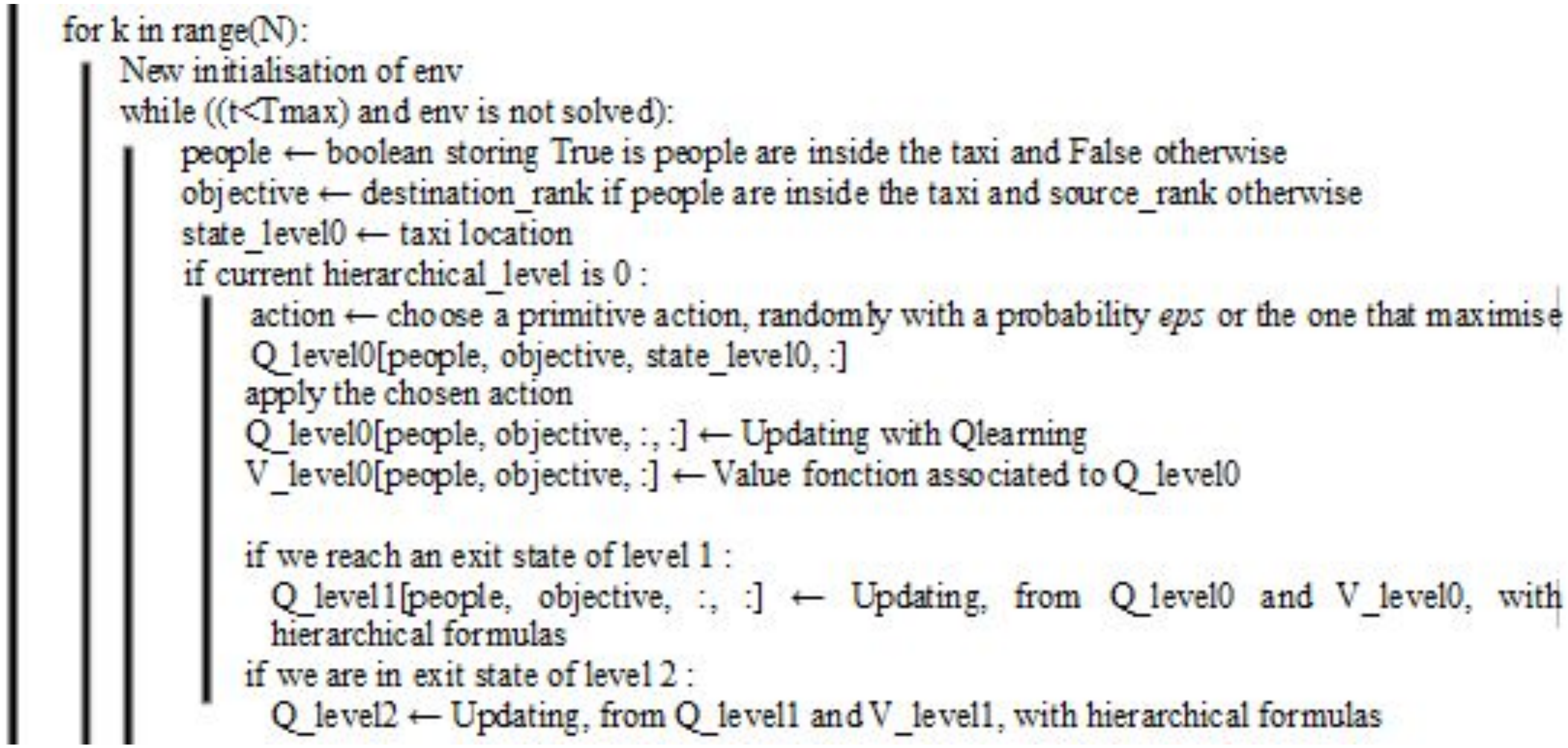
(a) Running average m_c of how often each concept c (taxi location, people presence in the cab) appeared as a peak at episode 50. Average over 30 runs of DDMQ

Implementation choices
Pseudo-code of HEXQ training



```
INPUT
eps = Exploration/exploitation factor
N = Number of epochs
Tmax = Time horizon

PROCESS
    #initialisation
    env ← Environment of the taxi problem
    Q_level0 ← random matrice of size (2, 4, 25, 6) such as $\forall (i, j, k, l)$, $Q\_level0[i, j, k, l] = -\infty$ if the action $l$ is
    not possible from the state $k$
    Q_level1 ← random matrice of size (4, 5, 8) such as $\forall (i, j, k)$, $Q\_level1[i, j, k] = -\infty$ if the action $k$ is not
    possible from the state $j$
    Q_level2 ← random matrice of size (4, 4) such as $\forall (i, j)$, Q_level2[i, j]=$-\infty$ if the action $j$ is not possible
    from the state $i$
```

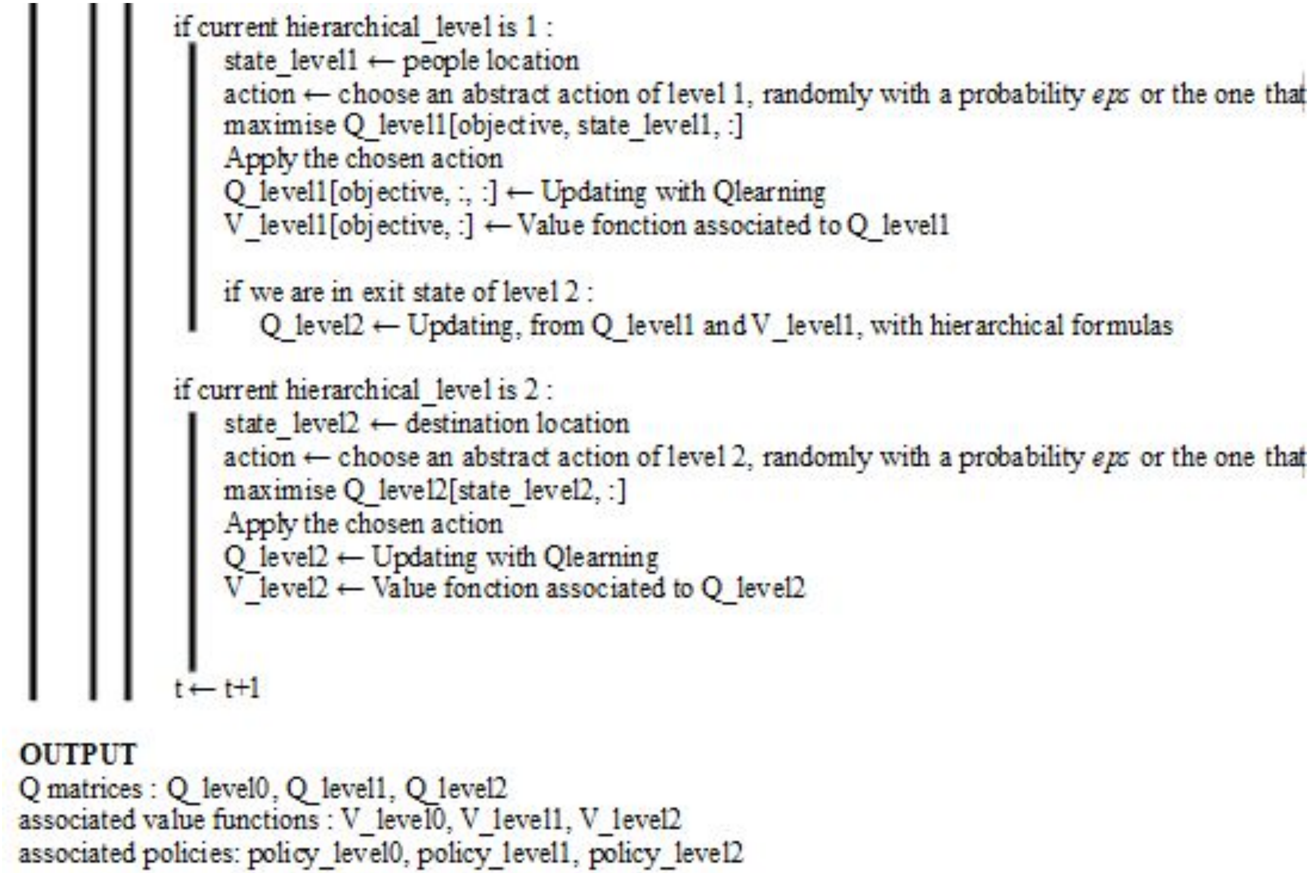
Implementation choices
Pseudo-code of HEXQ training



```
for k in range(N):
    New initialisation of env
    while ((t<Tmax) and env is not solved):
        people ← boolean storing True is people are inside the taxi and False otherwise
        objective ← destination_rank if people are inside the taxi and source_rank otherwise
        state_level0 ← taxi location
        if current hierarchical_level is 0 :
            action ← choose a primitive action, randomly with a probability eps or the one that maximise
             Q_level0[people, objective, state_level0, :]
            apply the chosen action
            Q_level0[people, objective, :, :] ← Updating with Qlearning
            V_level0[people, objective, :] ← Value fonction associated to Q_level0

            if we reach an exit state of level 1 :
              Q_level1[people, objective, :, :] ← Updating, from Q_level0 and V_level0, with
              hierarchical formulas
            if we are in exit state of level 2 :
              Q_level2 ← Updating, from Q_level1 and V_level1, with hierarchical formulas
```

Implementation choices

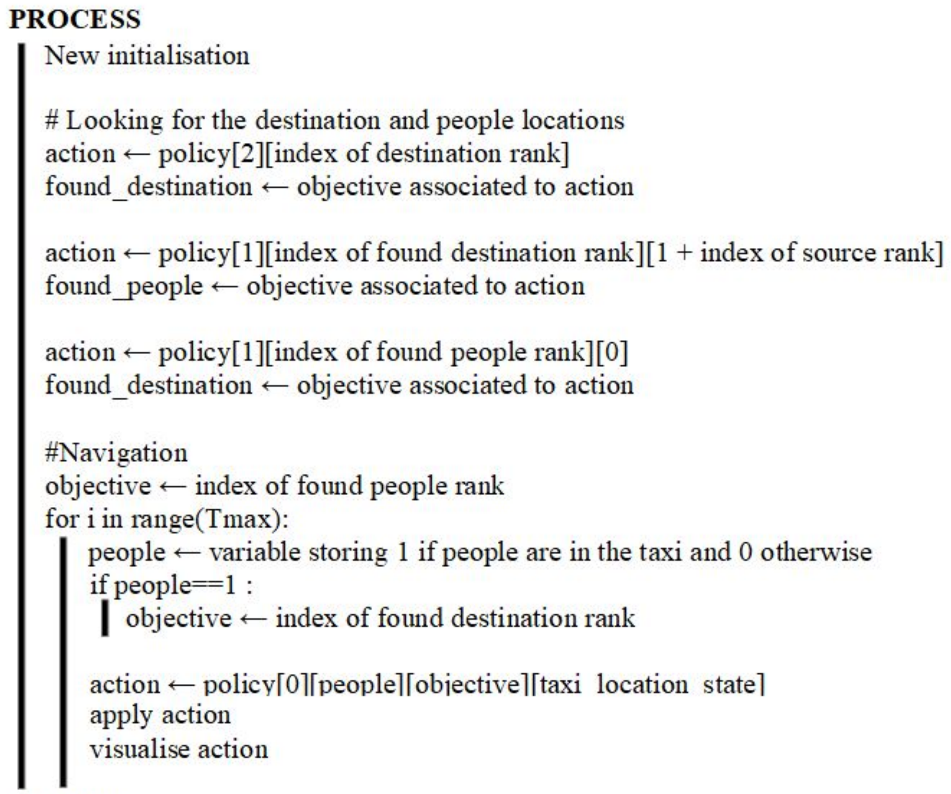
Pseudo-code of HEXQ training



```
            if current hierarchical_level is 1 :
                state_level1 ← people location
                action ← choose an abstract action of level 1, randomly with a probability *eps* or the one that
                maximise Q_level1[objective, state_level1, :]
                Apply the chosen action
                Q_level1[objective, :, :] ← Updating with Qlearning
                V_level1[objective, :] ← Value fonction associated to Q_level1

                if we are in exit state of level 2 :
                    Q_level2 ← Updating, from Q_level1 and V_level1, with hierarchical formulas

            if current hierarchical_level is 2 :
                state_level2 ← destination location
                action ← choose an abstract action of level 2, randomly with a probability *eps* or the one that
                maximise Q_level2[state_level2, :]
                Apply the chosen action
                Q_level2 ← Updating with Qlearning
                V_level2 ← Value fonction associated to Q_level2


            t ← t+1
```

OUTPUT
Q matrices : Q_level0, Q_level1, Q_level2
associated value functions : V_level0, V_level1, V_level2
associated policies: policy_level0, policy_level1, policy_level2

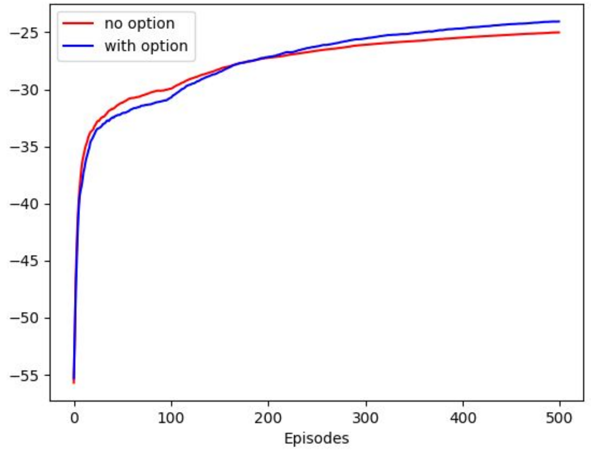
Implementation choices
Pseudo-code of HEXQ testing



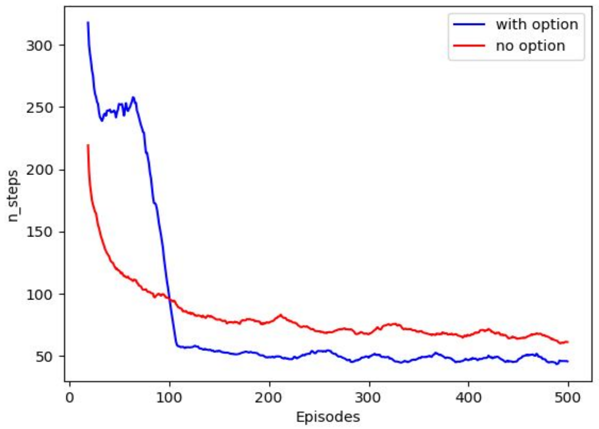
```
PROCESS
    New initialisation

    # Looking for the destination and people locations
    action ← policy[2][index of destination rank]
    found_destination ← objective associated to action

    action ← policy[1][index of found destination rank][1 + index of source rank]
    found_people ← objective associated to action

    action ← policy[1][index of found people rank][0]
    found_destination ← objective associated to action

    #Navigation
    objective ← index of found people rank
    for i in range(Tmax):
        people ← variable storing 1 if people are in the taxi and 0 otherwise
        if people==1 :
            objective ← index of found destination rank

        action ← policy[0][people][objective][taxi  location  state]
        apply action
        visualise action
```

Results - *DDMQ vs. basic Q-learning*

"task-relevant" metric
transition probability : 0.8



(b) For each algorithm, at a given episode, we plot the average of the discounted cumulative rewards across all previous episodes.



(c) Number of steps to reach the goal throughout the learning. Curbs are average over 30 runs for each of the two algorithms, and moving average over 20 episodes was then applied.