MAD
Decision Making

Learning
Complexe Behavior

Guillaume Lozenguez
@imt-nord-europe.fr

IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

1. **Back to Q-Learning on 421**
2. **Converging**
3. **Risky Game**
4. **Curse of Dimensionality**

1. **Back to Q-Learning on 421**
2. Converging
3. Risky Game
4. Curse of Dimensionality

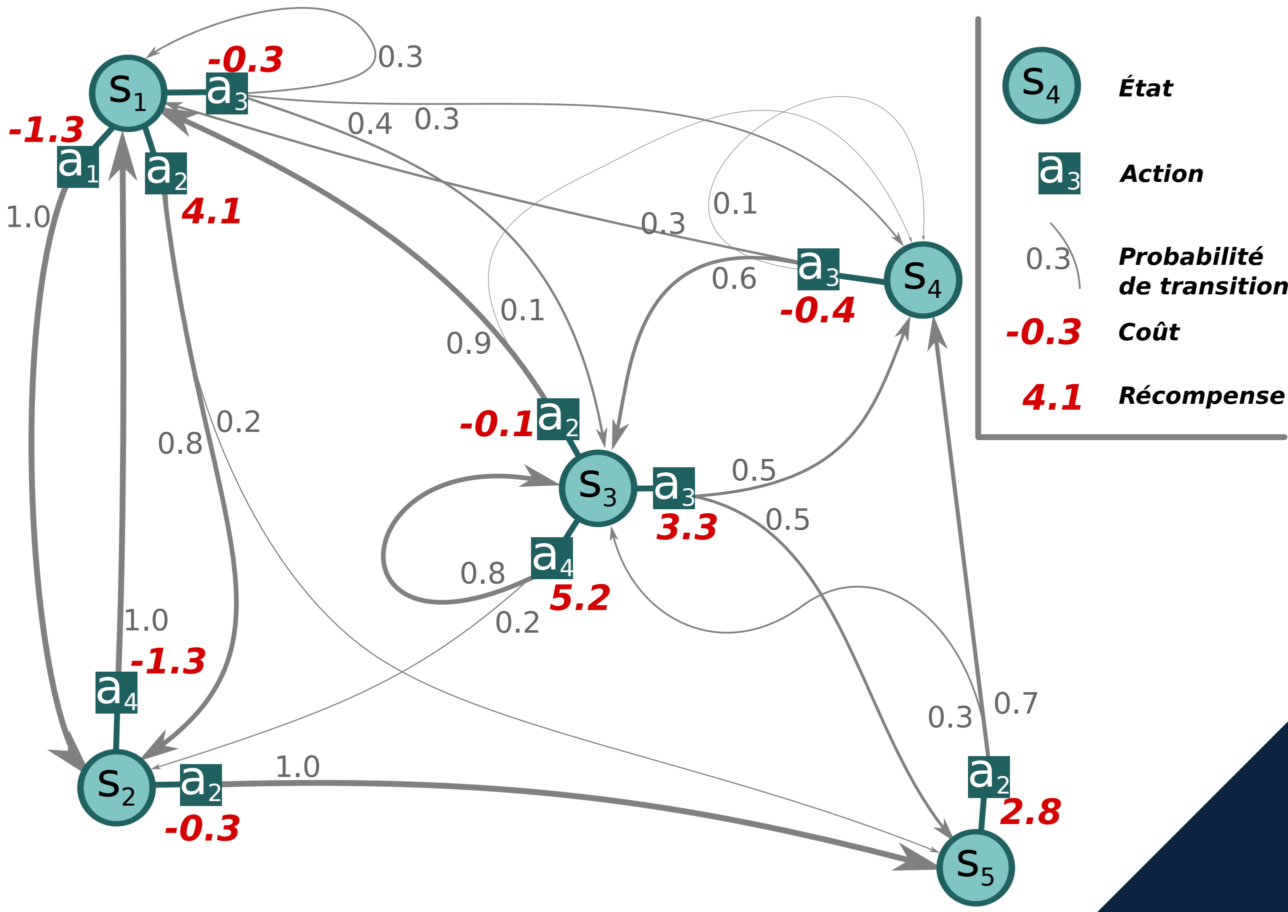
Hypotesis: it's Markovian

The system to control matches a Markov Decision Process

MDP: $\langle S, A, T, R \rangle$:

S : set of system's states
A : set of possible actions
T : S × A × S → [0, 1] : transitions
R : S × A → R : cost/rewards

We do have *S* and *A* but not *t* and *r*



Q-Learning: the basics

- Iterative update on (**state**, **action**) evaluation
- Q-Value equation:

$$Q(s^t, a) = (1 - \alpha)Q(s^t, a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s^{t+1}, a') \right)$$

- Few parameters:
  α learning rate ; ϵ Exploration-Exploitation ratio and γ discount factor.

Q-Learning: for instance

- Reaching *4-2-1* at *h-1* from *6-2-1* at *h-2* by doing *roll-keep-keep*.

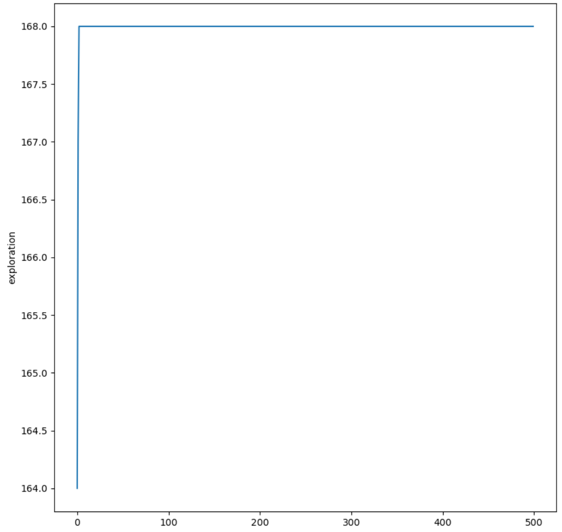
$$Q(\text{2-6-2-1},\ \text{r-k-k}) = (1-\alpha)Q(\text{2-6-2-1},\ \text{r-k-k}) + \alpha\left(r + \gamma \max_{a' \in A} Q(\text{1-4-2-1},\ a')\right)$$

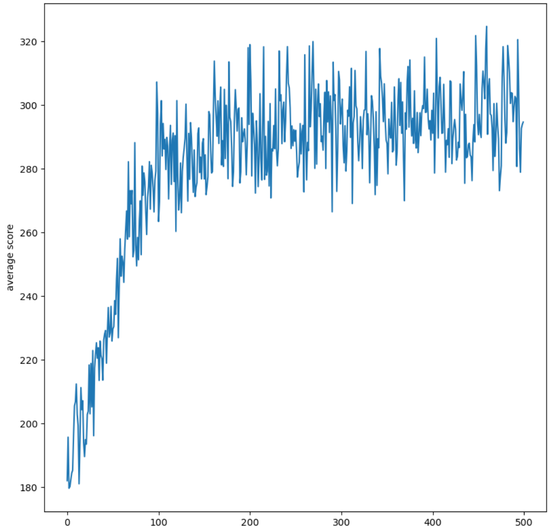
$$Q(\text{2-6-2-1},\ \text{r-k-k}) = (1-\alpha)\,40.0 + \alpha\,(0.0 + 80.0) \quad (a' = \text{keep}^3)$$

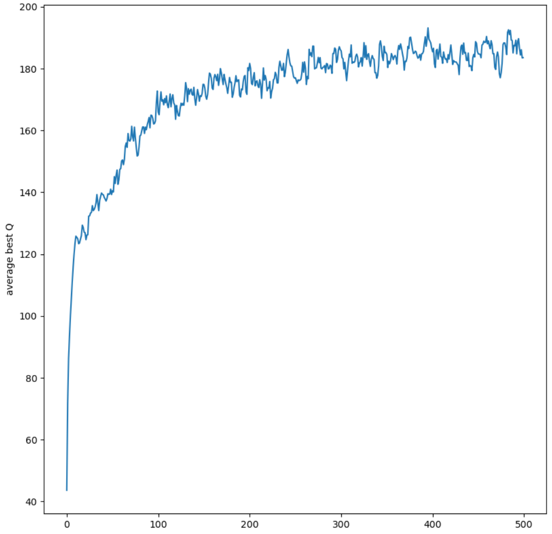
- With α learning rate at *0.1*, $Q(\text{2-6-2-1},\ \text{r-k-k})$ is now equals to *44*

Q-Learning: the basics

- With 500 steps of 500 games:







- α: 0.1 ; ϵ : 0.1 ; γ : 0.99 ;

Drawing plot in Python: pyplot

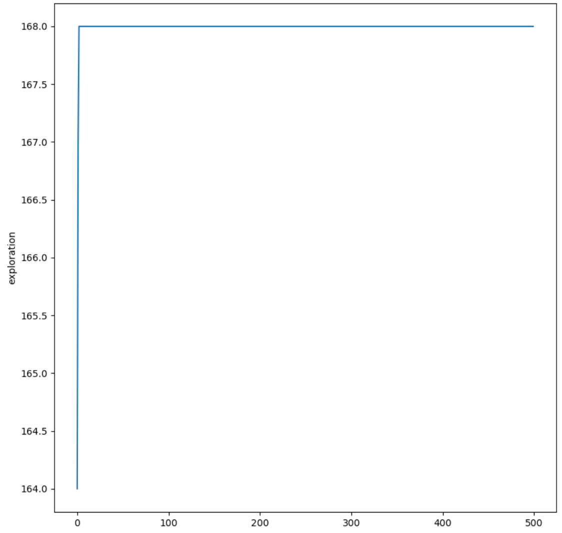
Codes:

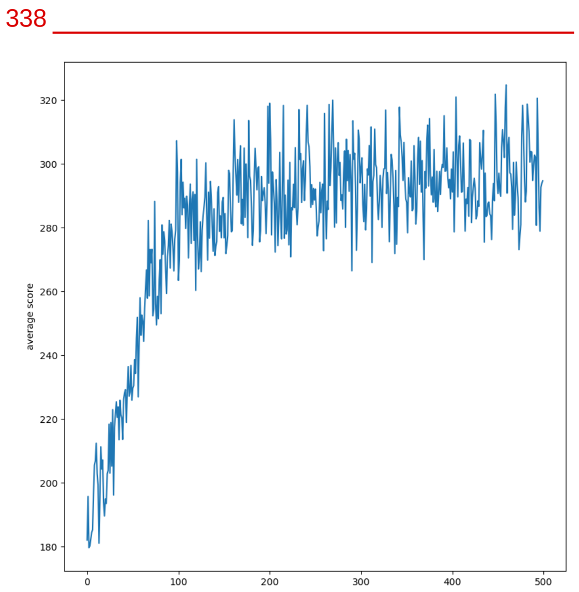
```
import matplotlib.pyplot as plt

..

plt.plot( values )
plt.ylabel( "mean of the y value" )
plt.show()
```

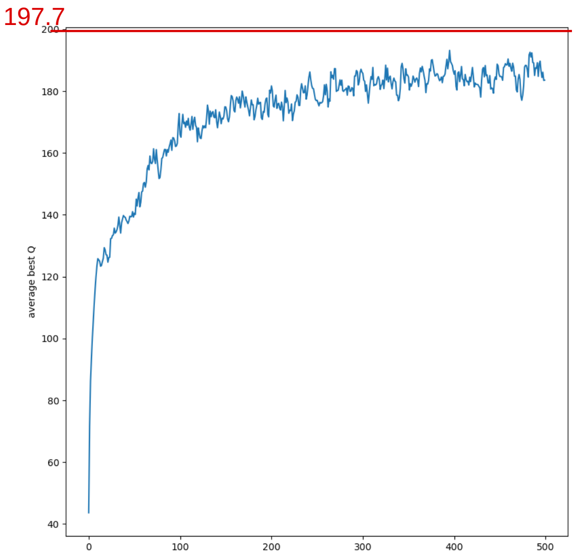
- Where `values` is a list of values in $\mathbb{R}$

Q-Learning: the basics

- With 500 steps of 500 games:



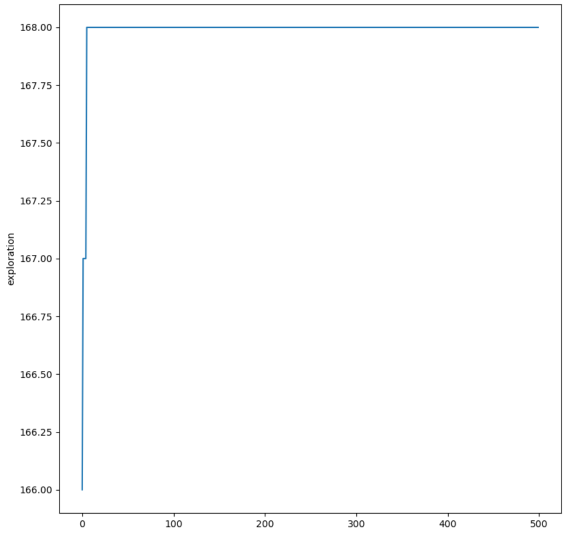


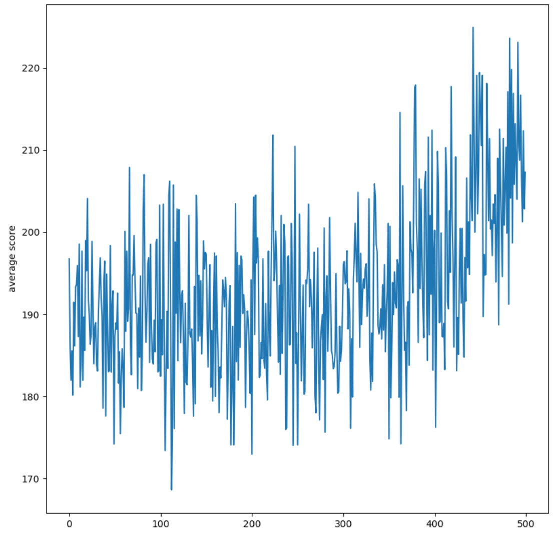


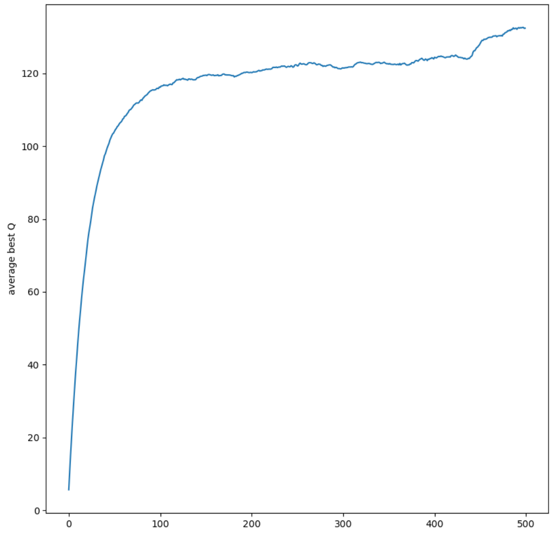
- With optimal threshold

Q-Learning: the basics

- With 500 steps of 500 games:



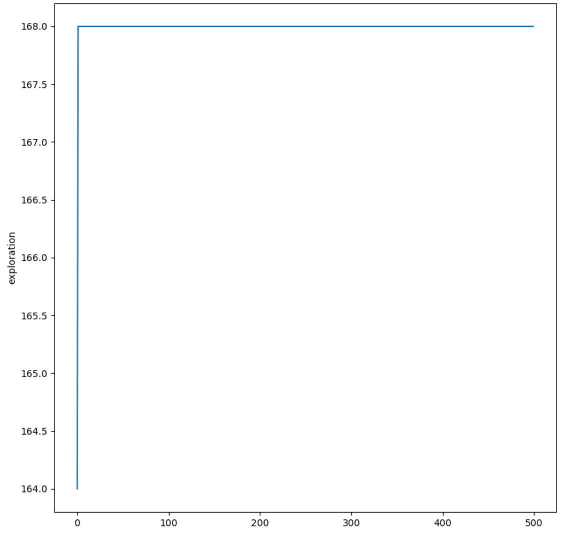


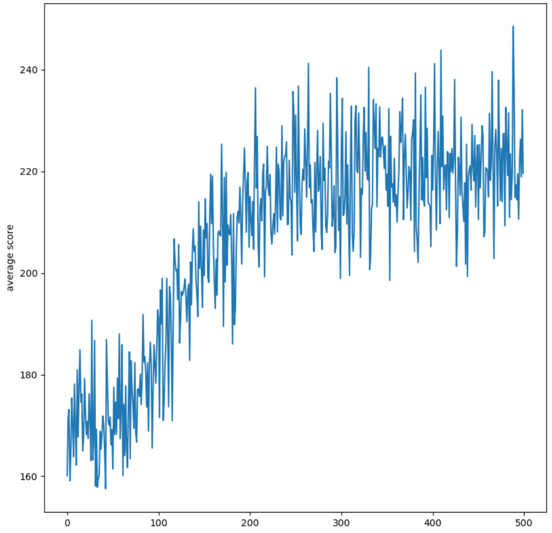


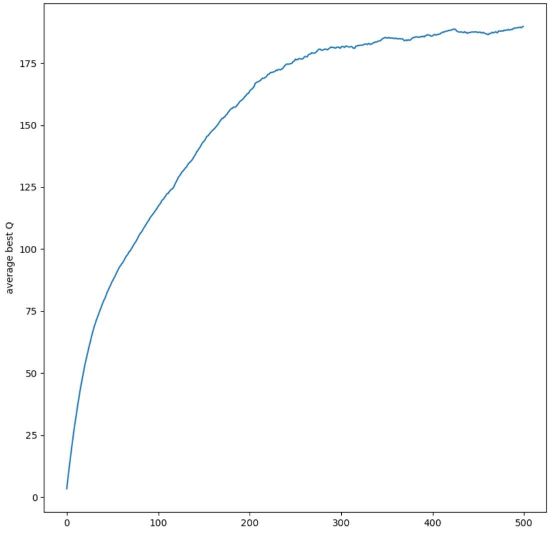
- α: 0.01 ; ϵ : 0.1 ; γ : 0.99 ;

Q-Learning: the basics

- With 500 steps of 500 games:







- α: 0.01 ; ϵ : 0.6 ; γ : 0.99 ;

Playing with the parameters:

- Generate rapidly "good" policies
- Converge on a maximal and stable Q-Values
  (an indicator for optimal policy)
- Potentially: be reactive to system modification (recovery)

Ideally: implement dynamic parameters

1. Back to Q-Learning on 421
2. **Converging**
3. Risky Game
4. Curse of Dimensionality

1. Back to Q-Learning on 421
2. **Converging**
3. Risky Game
4. Curse of Dimensionality

Converging

Cf. tutorial:

bitbucket.org/imt-mobisyst/hackagames/src/master/doc/index.md

1. Back to Q-Learning on 421
2. Converging
3. **Risky Game**
4. Curse of Dimensionality

1. Back to Q-Learning on 421
2. Converging
3. Risky Game
4. **Curse of Dimensionality**

The Curse of Dimensionality

1. **The Curse**
2. **Geometric Reduction**
3. **State Decomposition**

1. **The Curse of Dimensionality**
   - Example With 2 player 421
2. Geometric Reduction
3. State Decomposition

System Difficulty

Directly correlated to the state space:

The number of states: the Cartesian product of variable domains $|S|$ (minus some unreachable states)

- **421 game:** 3 dice-6 at the horizon 3: $\left(3 \times 6^3 = 648\right)$ but 168 effectives.

Then the branching:

Finally, the number of games:

System Difficulty

Directly correlated to the state space

Then the branching:

The number of possible actions and actions' outcomes.

- **421 game:** 2^3 actions, 6^r action outcomes (r, the number of rolled dice - *max. 3*).

Finally, the number of games:

The number of all possible succession of states until reaching an end.

$|Branching|^h$ (h the horizon) - Potentially $|S|^h$. **421 game:** $(6^3)^3$ games

Reminder over Combinatorics

With a Classical 32-card game: Possible distribution $32! = \quad 2.6 \times 10^{35}$

Human life: around 5×10^{7} seconds

Probability to play 2 times the same distribution in a human life is very close to 0

Learning 2-players-421

State space ?

Branching ?

First results..

Learning in Combinatorics Context

The root problem: handle large systems

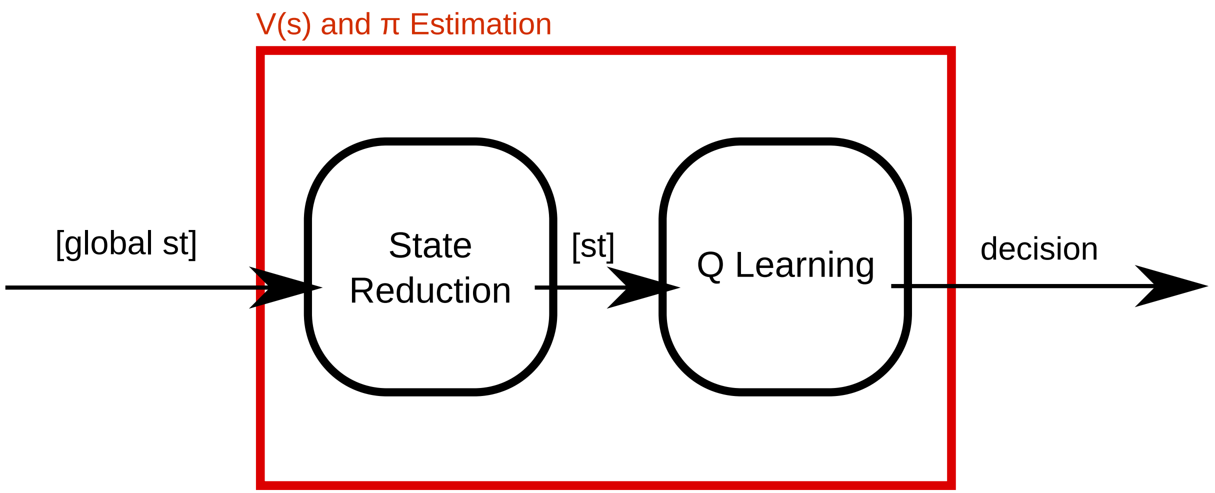
- Evaluate states $V : S \to \mathbb{R}$ or $Q : S \times A \to \mathbb{R}$
- Build a policy $\pi : S \to A$

A first basic solution:

- **Reduce the state space definition**

State reduction in QLearning

Project the states in a smallest space (dimension and size)

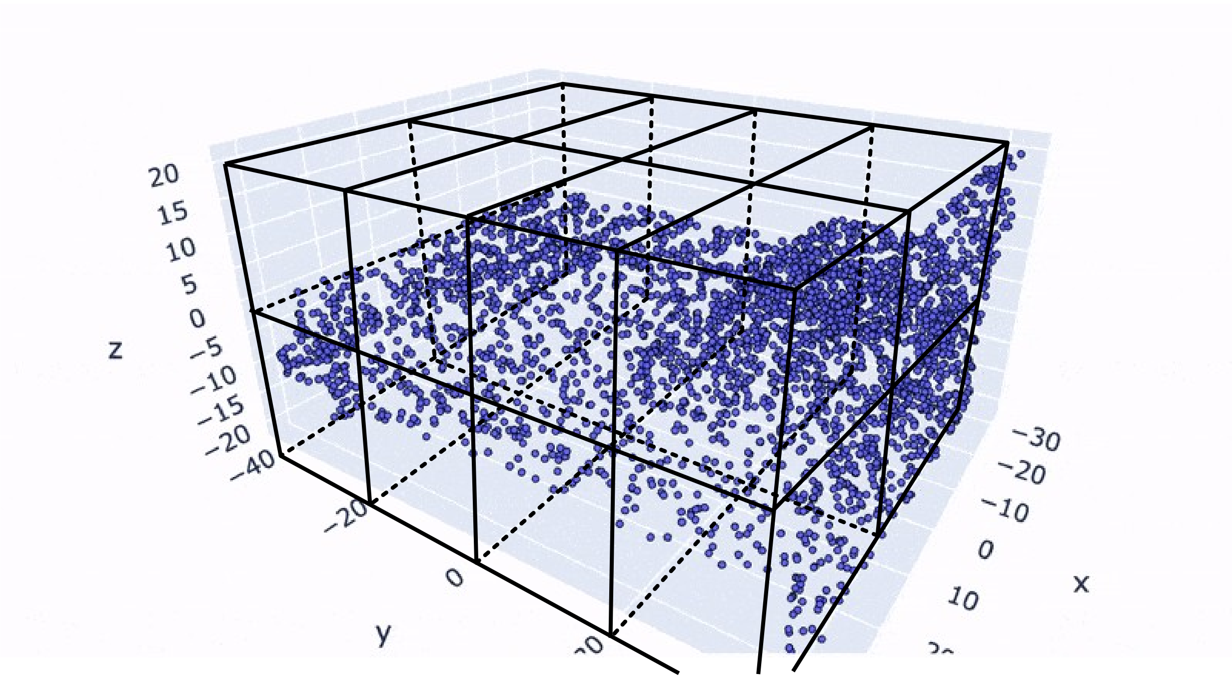


By mitigate the negative impact on the resulting built policy.

1. The Curse of Dimensionality
2. **Geometric Reduction**
   - Reduce the dimension (PCA)
   - Clustering (K-means)
3. State Decomposition

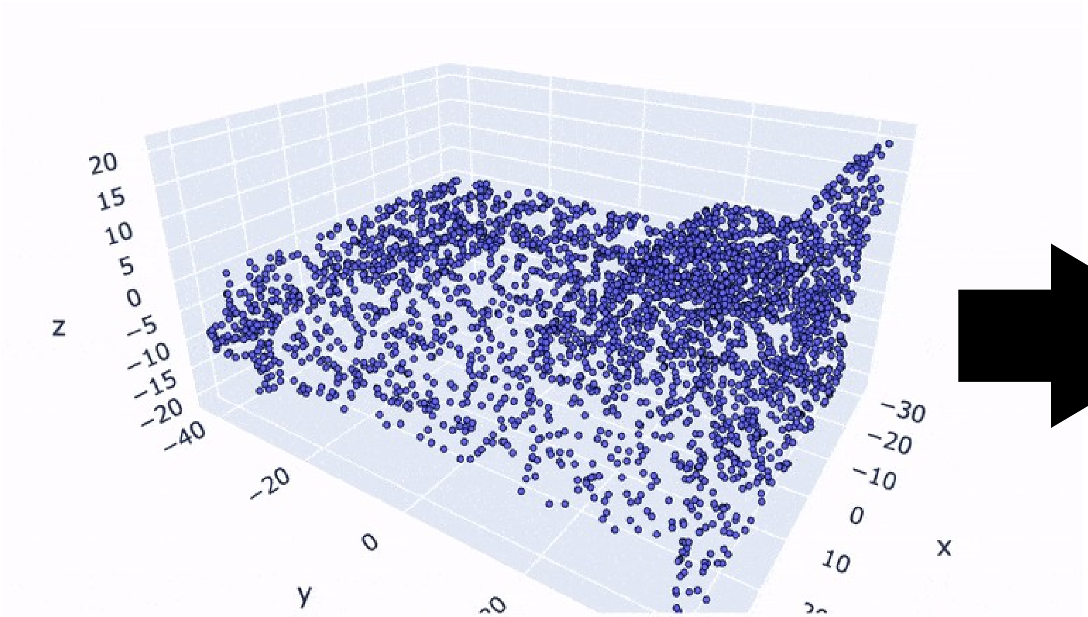
Geometry Reduction

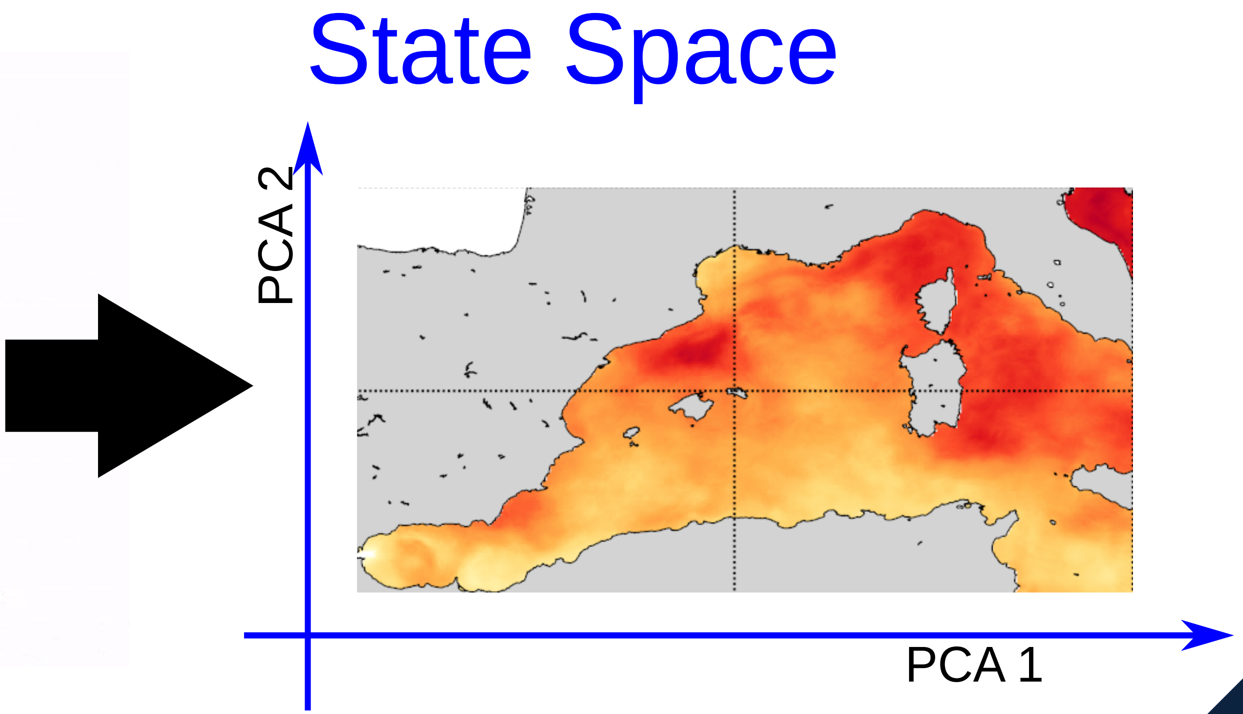
- Consider that **close** states are similar.
- Based on the assumption that: *it is possible to define a distance between States*
- By using regular discretization or adaptative clustering



Reduce the dimension - (Principal Component Analysis)

Searching the hyper-plan that better separate the data, in a given dimension.

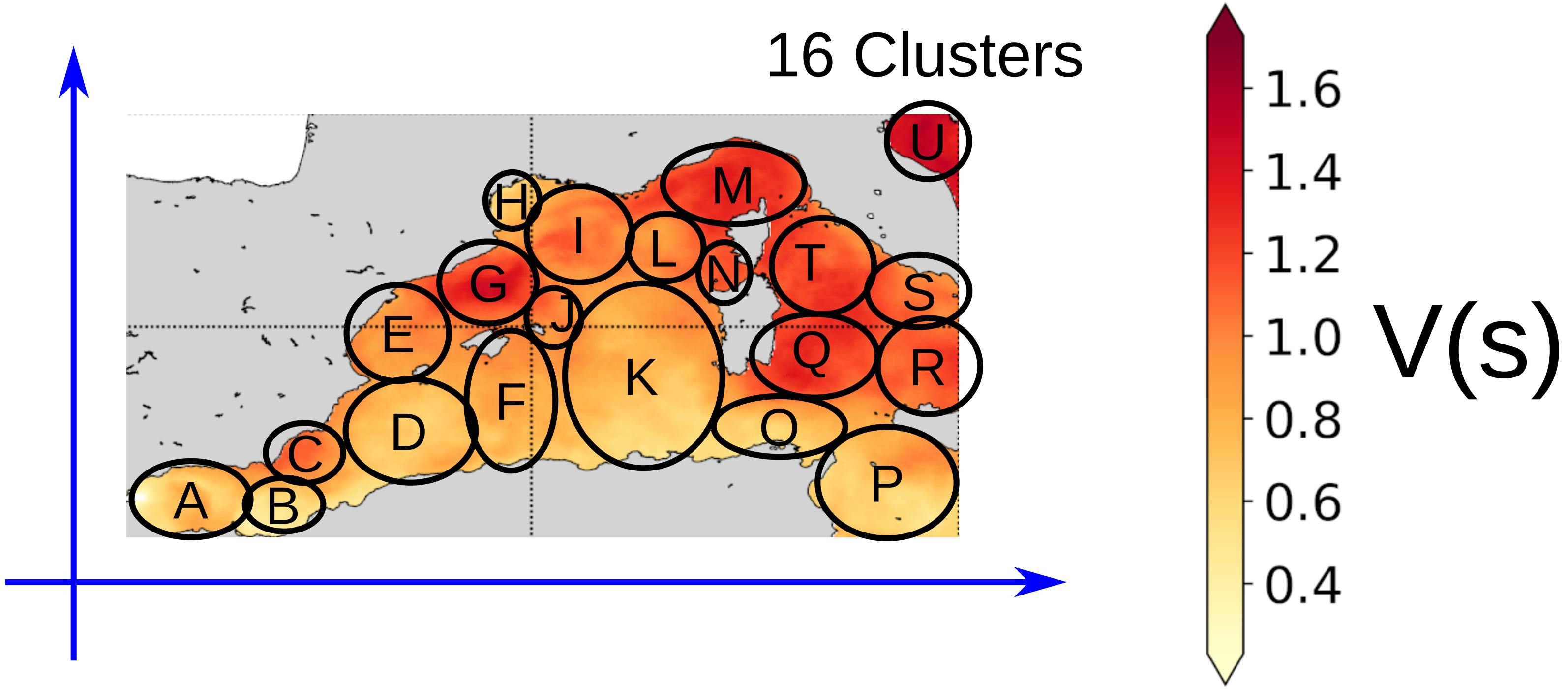




https://en.wikipedia.org/wiki/Principal_component_analysis

Clustering - (K-means)

regroup the states in coherent sets



K-means:
Searching the optimal *k* center positions that better group/separate the data

Basic 'simple' classification method

Principal Component Analysis (PCA)

Searching the hyper-plan that better separate the data, in a given dimension.

Python scikit-learn module: **sklearn.decomposition.PCA**

K-means

Searching the optimal *k* center positions that better group the data together.

Python scikit-learn module: **sklearn.cluster.KMeans**

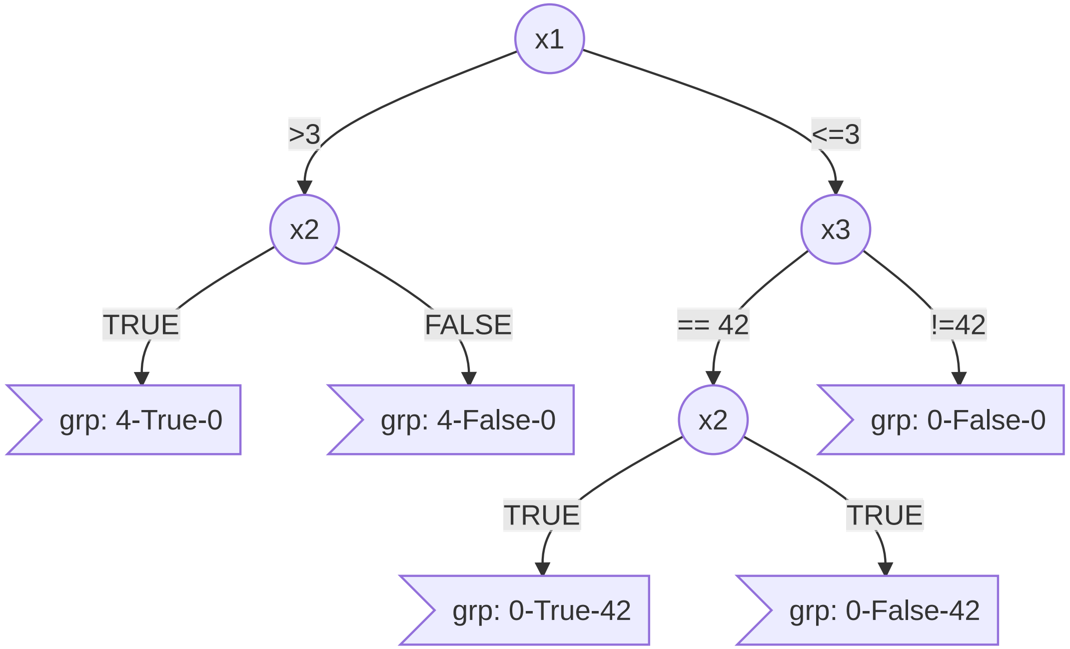
- Work well with 'linear state transitions' and different states density.
- Suppose a data set (trace) ideally with proper values

1. The Curse of Dimensionality
2. Geometric Reduction
3. **State-Space Decomposition**
   - Decision Tree
   - Example With 421

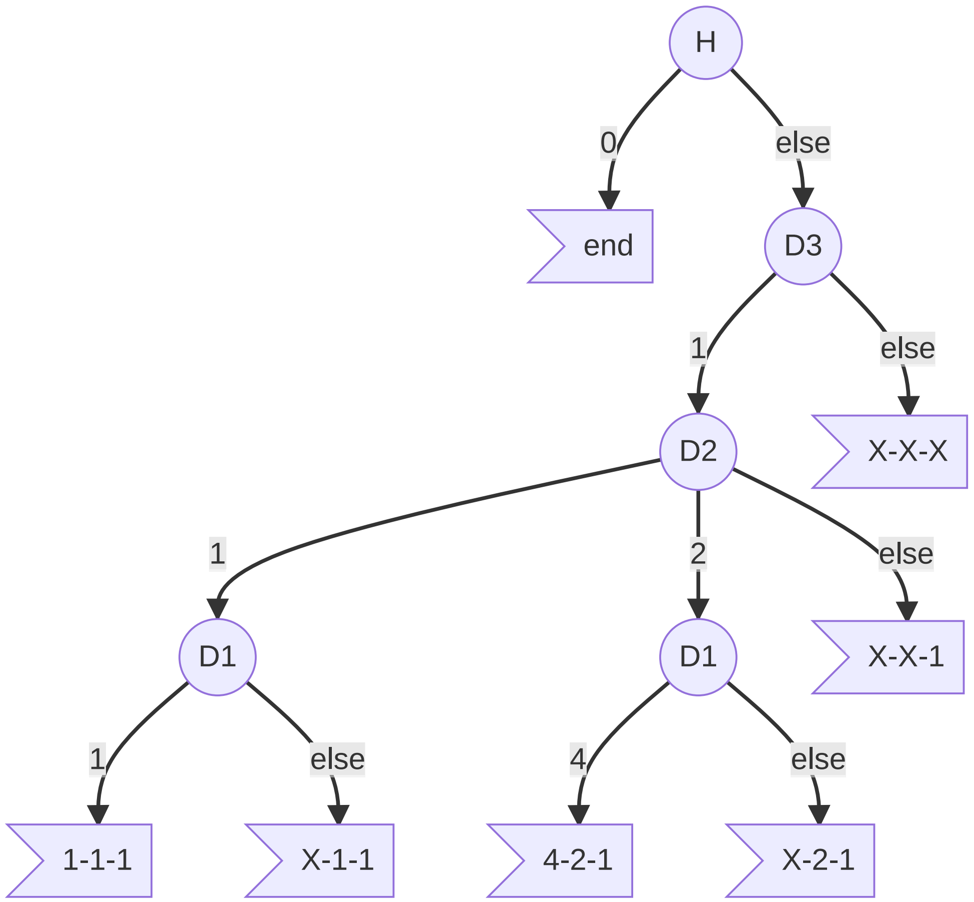
State-Space Decomposition

Factorized method: Based on state variable prevalence

- Decision tree (Again) **Nodes:** variables ; **Edges:** assignment ; **leaf:** group of states



Decision Tree On 421 Q-Learning



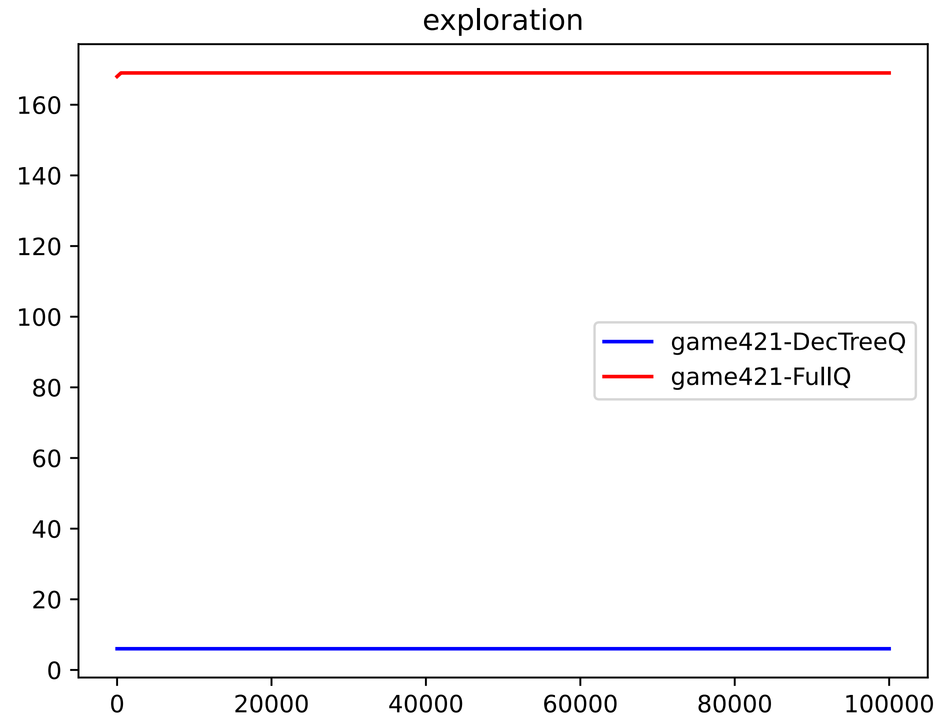
Decision Tree On 421 Q-Learning

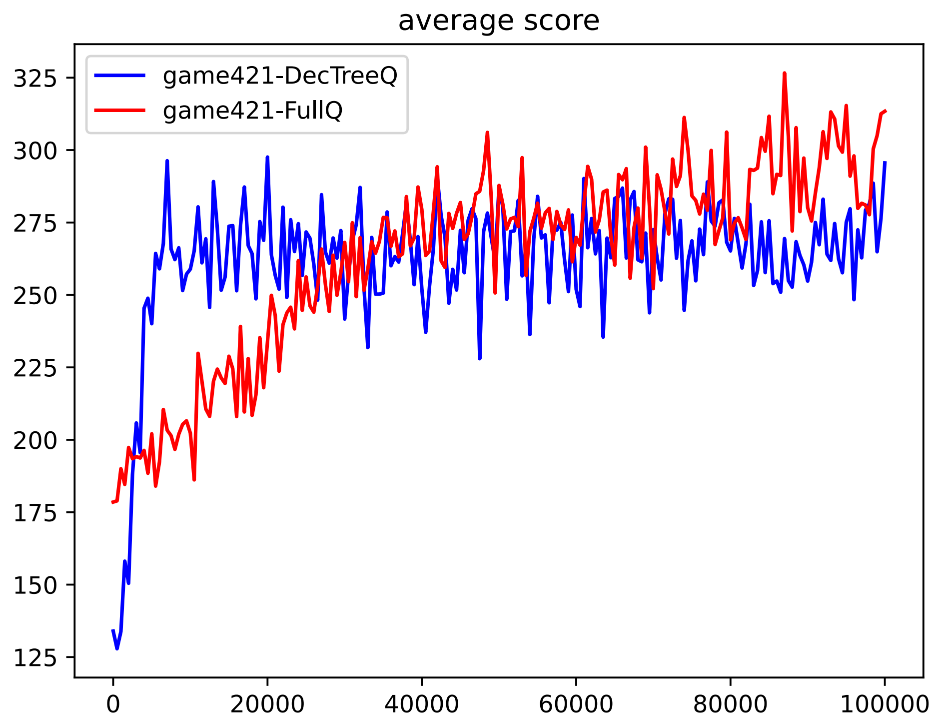
Simply reduce the state definition to *7* states..

```
def state(self):
   if self.turn == 0 :
      return 'end'
   if self.dices[2] == 1 :
      if self.dices[1] == 2 :
            if self.dices[0] == 4 :
               return "4-2-1"
            return "X-2-1"
      if self.dices[1] == 1 :
            return "X-1-1"
      return "X-X-1"
   return "X-X-X"
```

Decision Tree On 421 Q-Learning

Results:





Decision Tree Conclusion..

Conclusion:

It is all about defining the appropriate variable prevalence (Decision Tree Structure)

Learn the structure:

- Expert based Decision Trees or learned (ID3 algorithm)
- Again on python scikit learn: (module tree)

But..

The evaluation of the structure of the tree is performed by deadly execution of Q-Learning !