Bases de l'IA

Optimisation et résolution de problèmes

Isis TRUCK
Université Paris 8 / EPHE
2014-2015

Plan

- Optimisation ?
  - Exemple avec Dijkstra
- résolutions de problèmes
  - méthodes exactes et approchées ?
  - Exemple : le sac à dos
    - Résolution avec 3 méthodes
  - Apprentissage automatique
    - Plusieurs types d'apprentissage
    - Q-learning

2

Optimisation ?

- Optimisation : trouver la meilleure solution à **problème** en minimisant ou maximisant une *fonction objectif*
- Problème **difficile** *:* pour le résoudre, il n'est en général pas possible de fournir dans tous les cas des solutions en un temps raisonnable.
- Problème **NP-complet** : problème qu'on ne peut pas résoudre de façon optimale, sauf à étudier toutes les solutions une par une
- Par exemple, on peut vouloir trouver des **chemins optimaux** dans un réseau (composé de routeurs) : problème du **plus court chemin**

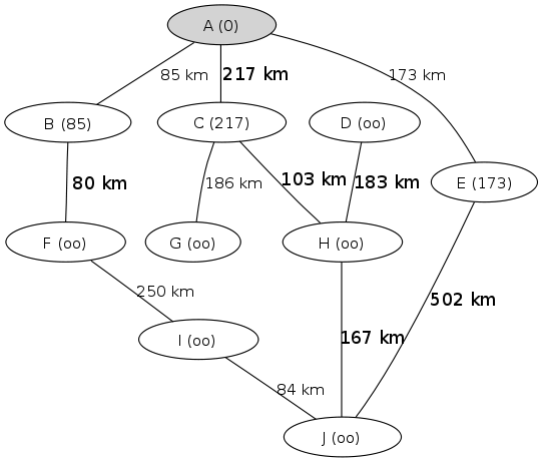
3

Chemins optimaux

- Une façon de résoudre le pb du plus court chemin est, par exemple, l'algorithme de **Dijkstra**
  - On modélise le réseau par un **graphe**
  - Chaque routeur est un **nœud**
  - Chaque liaison inter-routeur est un **arc**
  - Chaque arc est affecté d'un **poids** pour représenter le « coût » du passage du 1er nœud vers le 2e (coût = distance ; temps pour parcourir la distance ; argent à dépenser ; valeur reflétant l'encombrement du routeur, etc.)
  - A partir du point de départ, exploration systématique de toutes les solutions (des voisins directs), on prend la moins coûteuse, on marque le nœud suivant avec le coût cumulé. On choisit toujours le plus faible coût déjà calculé et on indique le nœud prédécesseur. On arrête quand tous les nœuds sont marqués; on part du nœud final et on remonte vers le nœud initial en utilisant le prédécesseur à chaque fois.

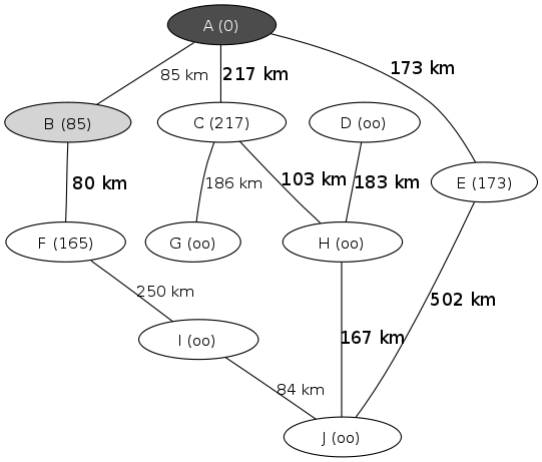
4

Dijkstra (1)



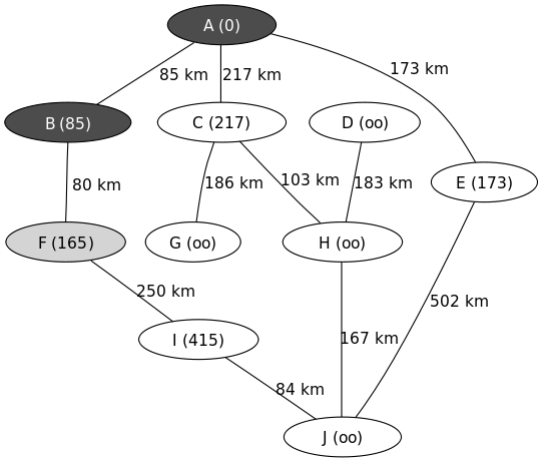
5

Dijkstra (2)



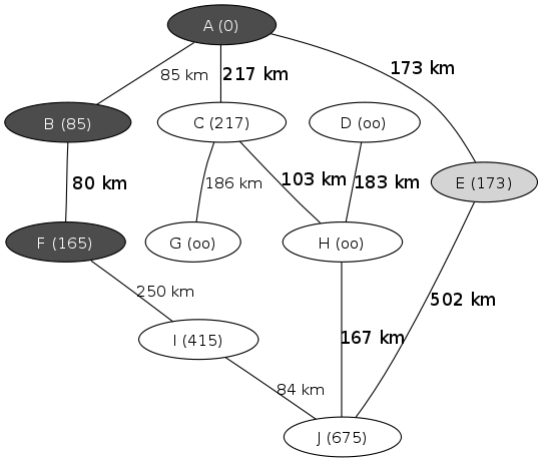
6

Dijkstra (3)



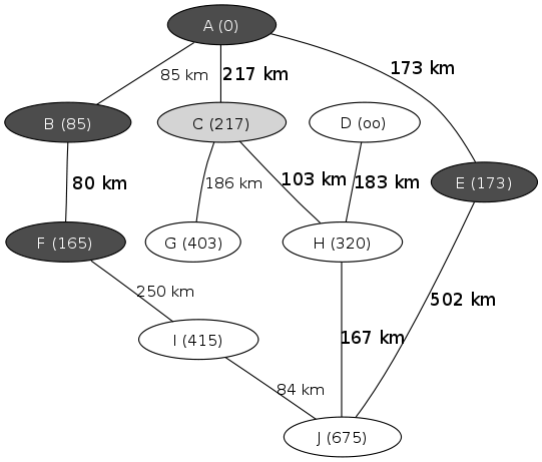
7

Dijkstra (4)



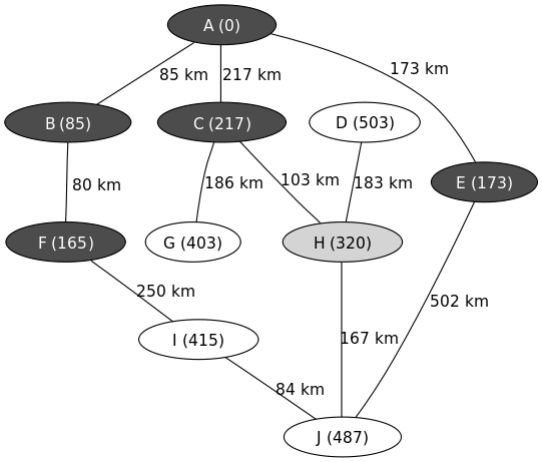
8

Dijkstra (5)



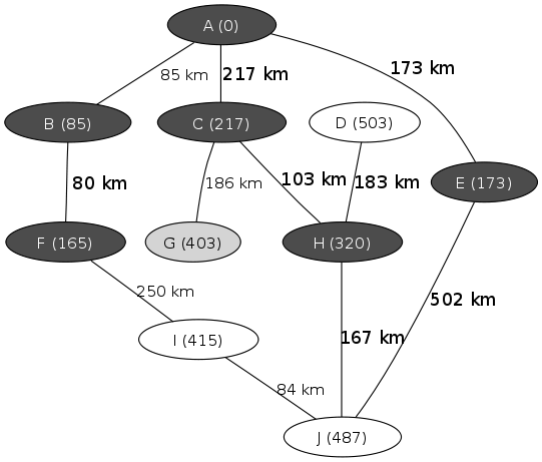
9

Dijkstra (6)



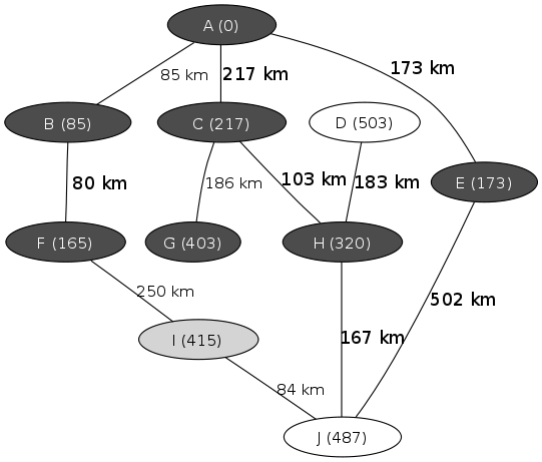
10

Dijkstra (7)



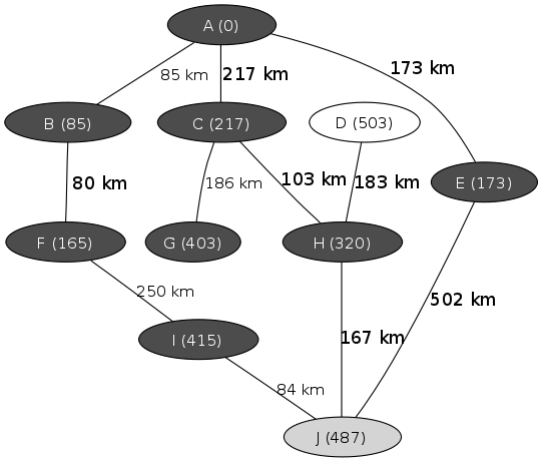
11

Dijkstra (8)



12

Dijkstra (9)



13

Résolution de problèmes

- Méthodes ?
  - **Exacte** : méthode permettant de trouver **la** bonne solution (solution optimale), mais les temps de calcul peuvent être très longs
  - **Approchée** : méthode permettant de trouver une solution satisfaisante, même si ce n'est pas la meilleure solution possible, et qui peut être atteinte dans un temps raisonnable (équilibre entre la qualité de la solution et le temps pour y parvenir)

14

Exemple du sac à dos

- Sac à dos : problème combinatoire, comme le problème du voyageur de commerce
- Principe : remplir un sac (d'une certaine taille, càd supportant un certain poids max) avec des objets qui ont un poids et un coût. Le coût est vu comme une valeur ou une **utilité**.
- But : remplir le sac « au mieux », c'est-à-dire mettre la plus grande somme (en valeur) dans le sac, sans dépasser le poids maximal
- Utilisation:
  - Cryptographie (allocation de ressources)
  - Gestion de personnel (le sac est l'effectif maximal de l'entreprise, objet = employé, coût= compétences de l'employé, poids = salaire et/ou disponibilité de l'employé)

15

Sac à dos : méthodes approchées (1)

- Méthode la plus simple : glouton
  - Appelée ainsi car elle veut réussir tout de suite : on choisit les maxima locaux en espérant que l'accumulation de maxima locaux donnera un maximum global
  - Principe : on part des objets qui ont le meilleur rapport coût-poids et on remplit le sac jusqu'à ce qu'il soit plein

16

Sac à dos : exemple

- Taille max du sac : 28 kg
- 14 objets :

Objet	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Valeurs	4	3	8	5	10	7	1	7	3	3	6	12	2	4
Poids	2	2	5	2	7	4	1	4	2	1	4	10	2	1

- Faire l'exercice en utilisant la méthode glouton
- Formaliser l'algorithme

17

Sac à dos : méthodes approchées (2)

- Algorithmes génétiques:
  - Utiles si méthode exacte inexistante ou si trop longue à trouver
  - Notion de sélection naturelle
  - Solution souvent acceptable avec des temps d'exécution corrects, même quand le problème comprend beaucoup de variables
  - On modélise le problème en considérant des individus identifiés par des chromosomes : **chaque chromosome est une solution potentielle**.
  - La population de départ est constituée de chromosomes qui ont été créés **au hasard**

18

Sac à dos : algorithme génétique (1)

- Chaque chromosome est constitué d'autant de gènes qu'il y a d'objets différents
- Un gène est codé sous forme binaire :
  - 1 : l'objet est dans le sac
  - 0 : l'objet n'est pas dans le sac
- On calcule l'efficacité de chaque individu (*fitness score*)
- Au temps suivant, on élimine le tiers (par ex.) des individus. Ils sont remplacés par de nouveaux, obtenus par **mutation** (par ex.), càd que l'on inverse les gènes des meilleurs chromosomes. On peut aussi les obtenir par *crossing over* :
- Recommencer

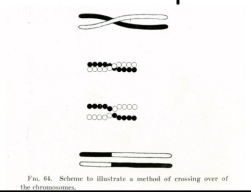


FIG. 64. Scheme to illustrate a method of crossing over of the chromosomes.

19

Sac à dos : algorithme génétique (2)

- Quels sont les choix à faire ?
  - Taille population à définir
  - Quantité d'individus à éliminer à chaque tour
  - Façon de procéder à la mutation et au *crossing over*
  - Question de la condition d'arrêt : quand est-on satisfait ?

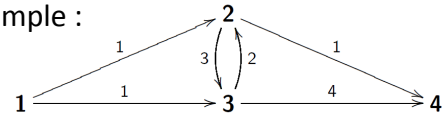
⇒ Faire l'exercice sur papier en considérant plusieurs « tours »

⇒ Quel algorithme ?

20

Sac à dos : méthodes exactes

- Par programmation dynamique
  - Suppose que le **principe d'optimalité** de Bellman est vérifié : « Une solution optimale pour le problème contient les solutions optimales pour tous les sous-problèmes »
  - C'est le cas dans le contrôle, en économie, en mathématique, en génie industriel… car les coûts/ bénéfices **s'additionnent** par prise de décisions successives.
  - Ce n'est pas le cas quand les sous-problèmes ne sont pas liés. Exemple :



    - Plus long chemin de 1 à 4 = 1,2,3,4
    - Plus long chemin de 1 à 2 (sous-pb du 1er) = 1,3,2

21

Sac à dos : progr. dynamique (1)

Principe

- On part du premier objet A.
  - On cherche les nouvelles combinaisons optimales (quelle utilité pour quel poids ?)
  - On enregistre ces nouvelles combinaisons comme des combinaisons déjà connues
- On continue avec le 2e objet B, mais en considérant toujours A
  - Même chose ensuite
- On arrête quand on a atteint le dernier objet

22

Sac à dos : progr. dynamique (2)

Mise en œuvre

- Soit M une matrice d'entiers, de dimension : nbObjets x (tailleMaxSac+1)

```
for (j=0; j < tailleMaxSac; j=j+1)
     if (poidsObjet[0] > j)
          M[0][j] = 0
     else M[0][j] = valeurObjet[0]
```

- Chaque « coefficient » de la matrice représente le bénéfice maximum possible pour les i premiers objets avec un poids j
- les autres lignes de M sont remplies par :

```
if (poidsObjet[i] > j)
  M[i][j] = M[i-1][j]
else
   M[i][j] = max(M[i-1][j], M[i-1][j-poidsObjet[i]] +
  valeurObjet[i])
```

23

Sac à dos : progr. dynamique (3)

- Exemple :

```
CONTENANCE DU SAC : 12 kg
- OBJET  : A| B|  C| D|  E|  F|  G| H
- POIDS  : 2| 3|  5| 2|  4|  6|  3| 1
- VALEUR : 5| 8| 14| 6| 13| 17| 10| 4
```

- Matrice M :

i\j	00	01	02	03	04	05	06	07	08	09	10	11	12
A	0	0	5	5	5	5	5	5	5	5	5	5	5
A/B	0	0	5	8	8	13	13	13	13	13	13	13	13
A/B/C	0	0	5	8	8	14	14	19	22	22	27	27	27
A/B/C/D	0	0	6	8	11	14	14	20	22	25	28	28	33
A/B/C/D/E	0	0	6	8	13	14	19	21	24	27	28	33	35
A/B/C/D/E/F	0	0	6	8	13	17	19	21	24	27	30	33	36
A/B/C/D/E/F/G	0	0	6	10	13	16	19	23	24	29	31	34	37
A/B/C/D/E/F/G/H	0	4	6	10	14	17	20	23	27	29	33	35	38

24

Sac à dos : prog. dynamique (4)

- Pour retrouver la liste des objets à mettre dans le sac :
  - on récupère dans la dernière ligne le poids minimal nécessaire pour faire le bénéfice optimal :

```
TANT QUE M[i][j] == M[i][j-1]   décrémente j
```

  - on récupère ensuite les objets :

```
TANT QUE j > 0 ET i > 0
   TANT QUE i > 0 ET M[i][j] == M[i-1][j]
      décrémente i
   j = j - PoidsObjet[i]
SI j >= 0
   Ajoute-objet(Objet[i])
décrémente i
```

Source : http://fr.openclassrooms.com/informatique/cours/introduction-a-la-programmation-dynamique/probleme-du-sac-a-dos

25

Sac à dos : progr. dynamique (5)

- Travailler sur l'exemple suivant :
  - Taille max du sac : 8 kg
  - 5 objets (utilité, poids) :
    - A (3,4)
    - B (5,3)
    - C (1,3)
    - D (2,2)
    - E (3,2)

Source : http://www.lirmm.fr/~sol/Rapports/RapportM1.pdf (pages 9 à 11)

26

Sac à dos : branch & bound

- Il s'agit d'une autre méthode exacte : la **séparation** et l'**évaluation**.
- L'idée est, bien que toutes les solutions puissent être énumérées, de **choisir** les solutions qu'on énumère grâce à certaines propriétés du problème, pour éviter d'énumérer des solutions menant à des échecs.

27

Apprentissage automatique

- Plusieurs types d'apprentissage
  - Supervisé :
    - On part d'une BD d'apprentissage contenant des couples (entrées, sorties) qui ont été déterminés par des experts (on a des classes de problèmes)
    - Le système, en voyant de nouvelles entrées inconnues, doit deviner les sorties adéquates, en fonction des couples existants dans la BD (il doit classer les entrées dans les classes existantes)
  - Non supervisé :
    - Le système doit apprendre seul à classer et à prendre une décision
    - Exemple : algorithme des K-moyennes
  - Par renforcement
    - Le système apprend un comportement suite à une observation (forme d'apprentissage non supervisé)
    - Le système produit sur l'environnement une valeur de retour qui guide l'apprentissage
    - Exemple : Q-learning

28

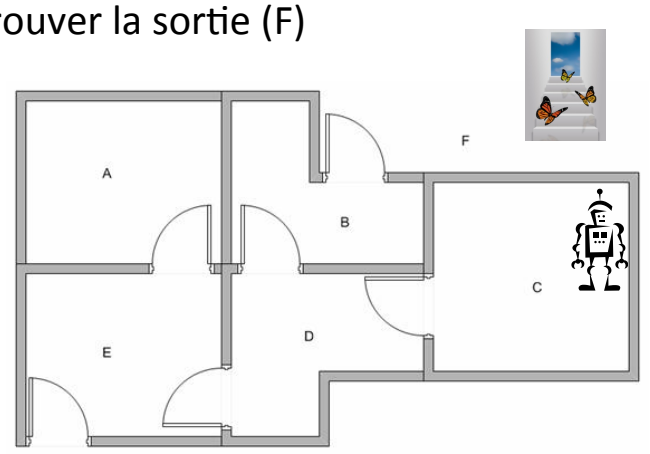
Q-learning (1)

- algorithme repose sur 3 types d'objets :
  - les états,
  - les actions (=transitions entre états)
  - et les récompenses (pour passer d'un état à un autre)
- Il s'agit de remplir des matrices en fonction notamment des états suivants.
- On calcule la **qualité** (d'où le Q) d'une combinaison « state-action »

29

Q-learning (2)

- Problème : 5 pièces => robot est dans C, il doit trouver la sortie (F)



30

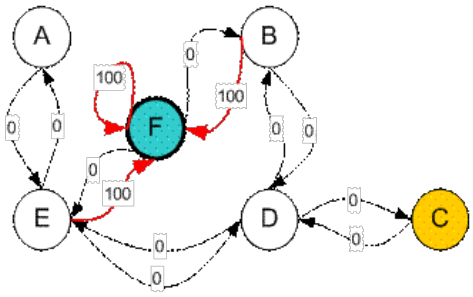
Q-learning (3)

- Modélisation sous forme de graphe :
  - État = pièce (A,B,C…F)
  - Action = passer d'une pièce à l'autre
  - Les pièces (états) sont les nœuds du graphe
  - Les déplacements du robot (actions) sont les arcs du graphe
  - Pour figurer qu'un passage existe entre 2 pièces, on dessine un arc en lui affectant initialement la valeur 0.
  - Une pièce menant directement au but se voit affecter la valeur 100 sur son arc.
  - Lorsqu'aucun passage existe, on ne crée pas d'arc.

31

Q-learning (4)

- Graphe (jaune : état initial ; bleu : état final)



Source : http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/
ou http://mnemstudio.org/path-finding-q-learning-tutorial.htm

32

Q-learning (5)

- Matrice :

R est fixe et sert aux calculs

action pour aller ds l'état :

Robot dans l'état	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

$$\mathbf{R} = \begin{array}{c} state \setminus action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} A \quad B \quad C \quad D \quad E \quad F \\ \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

- Hyp. (pour simplifier) : Robot sait qu'il y a 6 états

33

Q-learning (6)

- Règle de transition :
  $\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$
- Il y a 2 matrices : Q (matrice nulle, au départ) et R
- γ est un paramètre d'apprentissage (par exemple 0.8) et Max [Q(*next state, all actions*)] est la plus grande valeur de Q entre l'état suivant et toutes les valeurs possibles
- Robot va apprendre au travers de l'expérience sans supervision.
- Il va tout explorer, état par état, jusqu'à ce qu'il atteigne la sortie (F)
- Chaque exploration est un **épisode**.
- Il y aura un épisode où le robot atteindra la sortie (convergence)

34

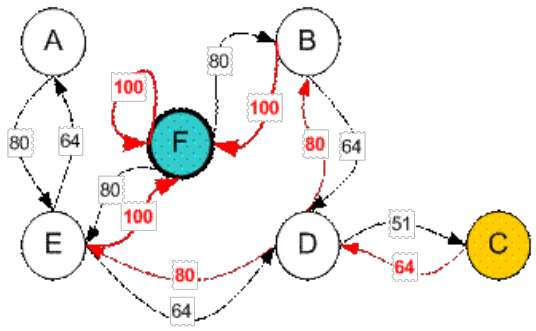
Q-learning (7)

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad \mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\mathbf{Q} = \begin{array}{c} state \setminus action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 400 & - \\ - & - & - & 320 & - & 500 \\ - & - & - & 320 & - & - \\ - & 400 & 256 & - & 400 & - \\ 320 & - & - & 320 & - & 500 \\ - & 400 & - & - & 400 & 500 \end{bmatrix} \end{array} \qquad \tilde{\mathbf{Q}} = \begin{array}{c} state \setminus action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 80 & - \\ - & - & - & 64 & - & 100 \\ - & - & - & 64 & - & - \\ - & 80 & 51 & - & 80 & - \\ 64 & - & - & 64 & - & 100 \\ - & 80 & - & - & 80 & 100 \end{bmatrix} \end{array}$$

Normalisation : division par 5

Q-learning (8)



- Solution : C – D – B – F
- A tester avec un autre exemple (autres chiffres)