Winner Determination

Tabu Search / Q learning

Aoucher Kenza

aoukenza@gmail.com

Benmessaoud Ahmed Sif

ahmed.sif.benmessaoud.13@gmail.com

Table des matières

Liste des figures

I Introduction

1 Formulation du problème

Le problème du détermination du gagnant dans les enchères combinatoires (*WDP : winning determonation problem*) et un problème qui a été catégorisé comme étant NP-complet [1]. Une instance du problème est représentée par un ensemble I de m objets et un ensemble B de n enchères, tel qu'une enchère est un sous ensemble d'objets et une valeur qui représente le prix ou l'offre. Une solution du problème est un ensemble W d'enchères gagnantes de taille p.
Le problème revint alors à maximiser les gains, tout en évitant les conflits sur les objets.

$$Maximize \sum_{i}^{p} B_i \tag{I.1}$$

avec

$$\nexists B_i, B_j \in W / items(B_i) \cap items(B_j) \neq \emptyset \tag{I.2}$$

2 Génération d'instance

Nous rappelons qu'une instance de ce problème est un ensemble d'objet ainsi qu'un ensemble d'enchères sur des sous ensembles d'objets accompagné d'une valeur.
Les paramètres à définir afin de créer une instance sont :

1. nombre d'objets
2. nombre d'enchères
3. nombre minimal et maximal d'objets par enchère

4. valeur maximale

On génère ainsi le nombre d'enchères souhaité avec un nombre aléatoire (entre min et max) d'objets aléatoires, avec une valeur aléatoire (de max/2 à max).

3 Algorithme *brute force*

L'approche force brute revient à tester toutes les possibilités, les comparer et trouver la meilleure. L'algorithme implémenté est récursif et est présenté ci-dessous :

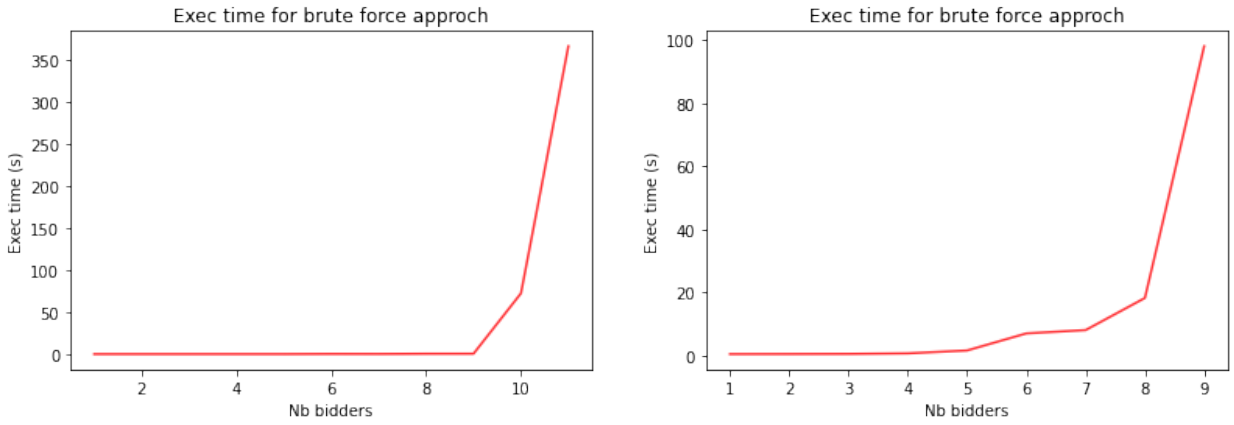
```
initialize possible solutions as an empty list


BruteForce(current gain, available bidders, available items)
    for each available bidder b
        if b.items in available items
            remove b from available bidders
            remove b.items from available items
            BruteForce(current gain + b.price, available bidders,
             ↪  available items)
    add solution to possible solutions


choose max solution in possible solutions
```

Complexité

Comme le problème est NP-complet la complexité est exponentielle, les figures 1a et 1b montrent l'évolution du temps d'exécution en fonction du nombre d'enchères sur différentes instances.



(a) Nombre d'objet=100, (min, max)=(2, 5)

(b) Nombre d'objet=1000, (min, max)=(5, 10)

FIGURE 1. – Démonstration de la complexité expérimentale de la force brute

Impacte du ratio du nombre de objet par enchère

Le ratio entre le nombre d'objets total et le nombre d'objet par enchère influe sur la distribution du conflit entre objets. En effet, si le nombre d'objet est m un nombre assez proche de x augmentera la probabilité d'avoir un conflit entre 2 enchères données. A l'inverse, un petit nombre d'objets par enchère minimise le risque d'obtenir un conflit. Ceci impacte le nombre d'appels récursif nécessaire de sorte qu'il est anti-proportionnel au nombre de conflits. La figure 2 montre l'évolution du temps d'exécution en fonction du nombre d'enchères sur 3 instances de 4600, 120 et 100 objets par enchères, le nombre d'objets max est fixé à 5000.

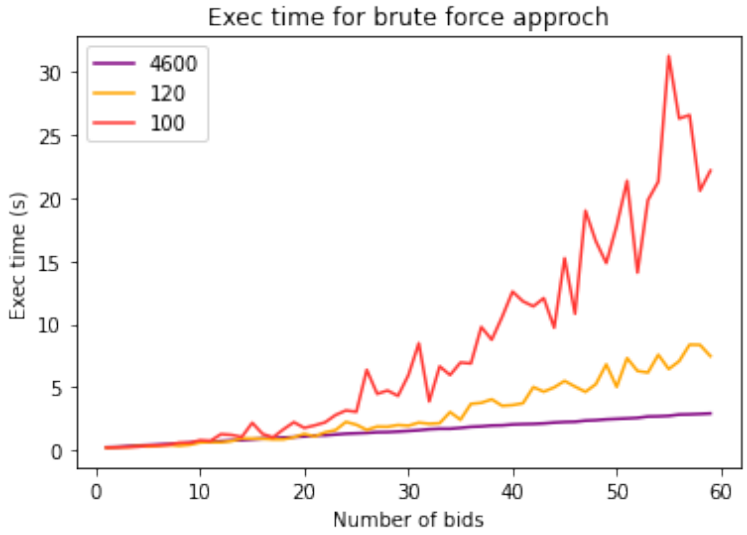


FIGURE 2. – Influence du nombre d'objet par enchère sur le temps d'exécution

4 Bref aperçu sur le *q learning*

Le q learning est une méthode de résolution très récente, elle commence à peine à être appliqué et utilisé sur les problèmes combinatoires classiques tel que le *Knapsack Problem* [2, 3].

Algorithm Q-Learning Algorithm

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$ for all $s \in S^+, a \in A(s)$ as 0
for each episode **do**:
    Initialize s
    for each step of episode **do**
        Choose a from s using an $\varepsilon - greedy$ policy derived from Q
        Take action a, observe r, s'
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$
        $S \leftarrow S'$
        if S is a terminal state **then**
            break
        end if
    end for
end for

FIGURE 3. – Algorithme du *Q-learning* [4]

5 *Tabu search*

Algorithm TS Algorithm

Initialize list of possible actions *ActionList*
Initialize the length of the list of actions *nAction*
Initialize the length of the Tabu List
Generate a random initial solution and label it as the current best solution
Initialize the Tabu List with all zeros *TC*
Initialize the number of iterations *MaxIt*
for j in range 0 to *MaxIt* **do**
    for i in range 0 to *nAction* **do**
        if $TC_i = 0$ **then**
            Perform an action *i* from the *ActionList*
            Store the new solution and action index *i*
            if New solution $\geq$ Current best solution **then**
                Store the best new solution as a new variable
                Store the best new solution
                Store the index of this best new action
            end if
        end if
        Set the current solution as current optimal value
        Store the index of the best action taken
        Update the Tabu list
        if Current optimal value estimate $>$ Previous optimal value estimate **then**
            Store the current optimal value as the current global solution estimate
        end if
    end for
end for

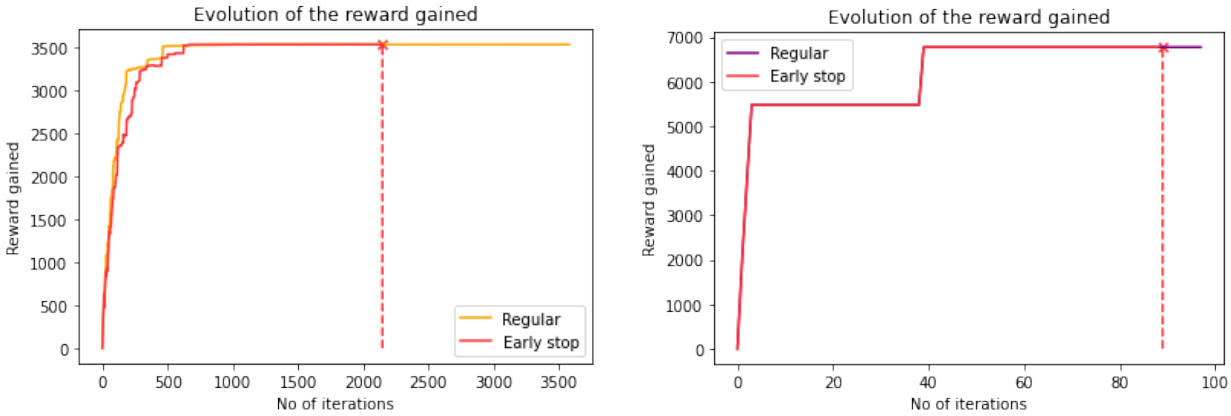
FIGURE 4. – Algorithme du *tabu search* [4]

II Méthodes et résultats

La valeur de agamma pour le *Q learning* est fixée à 0.4 après une série de testes.

1 Arrêt prématuré

Afin d'optimiser l'exécution nous implémentons un *early stop* lors du remplissage de la Q matrice. La figure 5 montre qu'a partir de l'itération 2000 la récompense atteinte est constante il n'est donc pas nécessaire de continuer. Lors de l'exécution avec l'arrêt prématuré stoppe la boucle d'entraînement, comme le montre la figure.



(a) *Q learning* **(b)** *Tabu search*

FIGURE 5. – Impact de l'arrêt prématuré

La figure 6 montre que le choix de la valeur du seuil d'arrêt est trés important, en effet pour cette exécution le modèle s'arrête juste avant de trouver une meilleure solution. Dans ce cas il n'atteindra jamais le gain optimal. Cette valeur est alors trouvée par l'expérimentation.

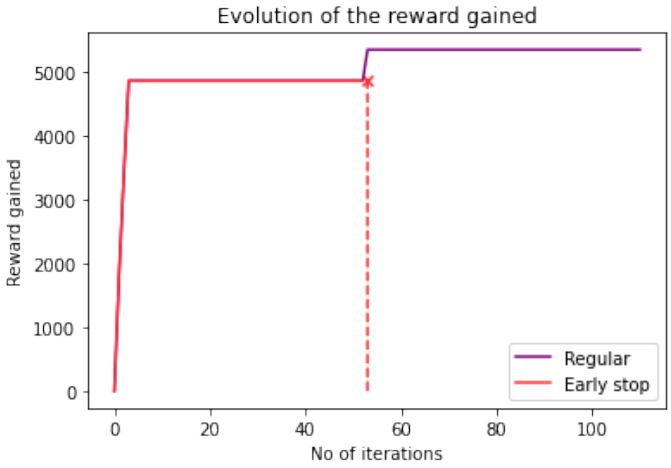


FIGURE 6. – Mauvais choix de la valeur de l'*early stop*

2 Validation des méthodes

Nous savons déjà que la force brute atteint la solution optimale du problème car elle teste toutes les possibilités. Afin de vérifier que notre méthode est valide il suffit de comparer ses résultats avec ceux de la force brute.

Nb enchères	BF		QL		TS	
	Temps	Gain	Temps	Gain	Temps	Gain
2	0.02	2693	2.41	2693	0.16	2693
4	0.23	5764	4.11	5764	0.24	5764
6	0.48	6204	5.41	5024	0.29	6204
7	7.81	9416	10.62	9416	0.30	9416
8	8.49	9457	20.37	8447	0.28	8447
9	436.95	11114	17.13	11114	0.34	11114

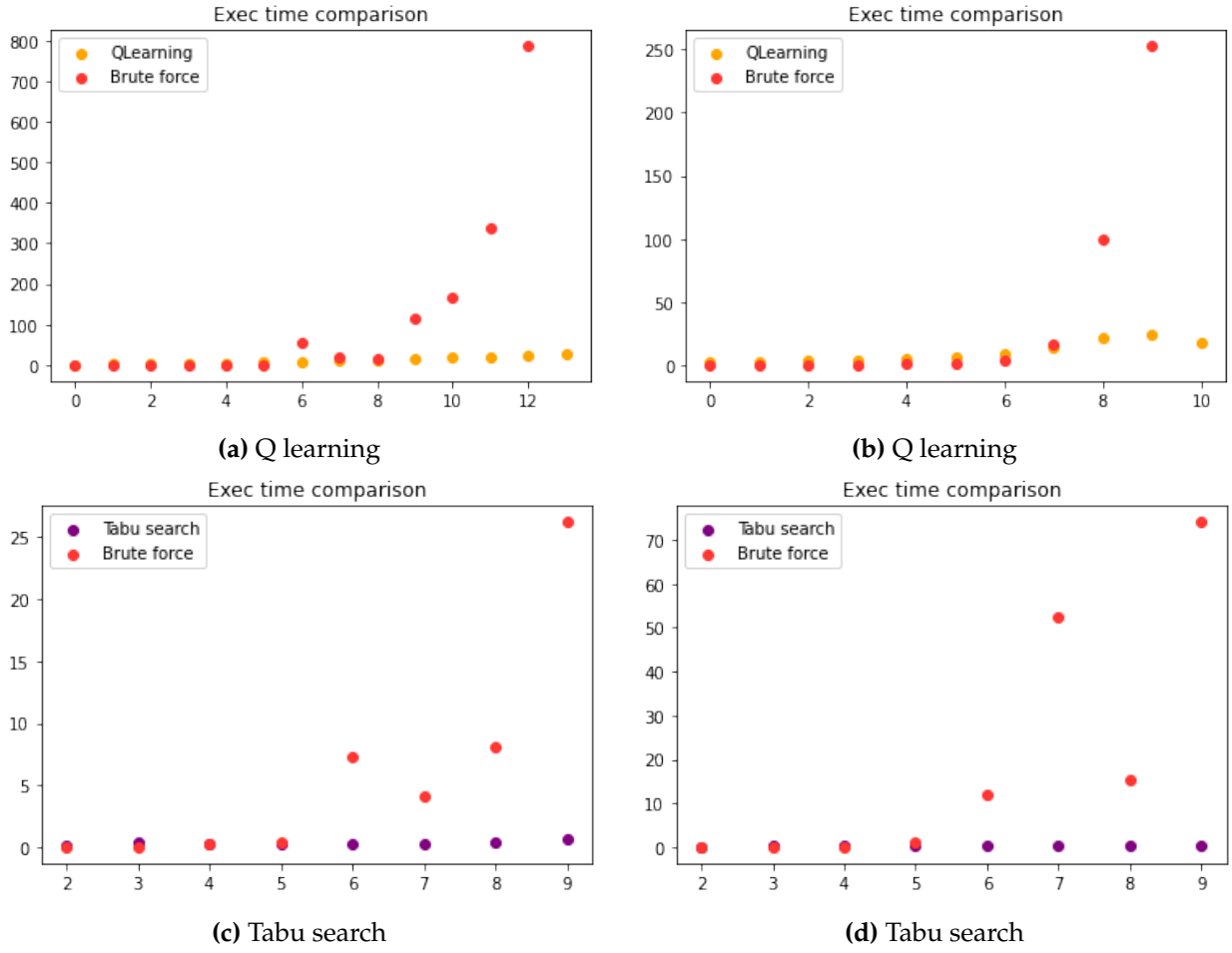
TABLE 1. – Instance de 1000 objets et 10 objets par enchère

Analyse

Le tableau 1 montre que la méthode *Q-learning* implémentée est valide et trouve dans la majorité des cas la solution optimale, sinon atteint un gain proche du meilleur. L'algorithme *tabu search* atteint la solution optimale dans pratiquement l'ensemble des cas.

3 Comparaison du temps d'exécution

Les exécutions suivantes se font sur des instances de 1000 objets et 10 par enchère.



(a) Q learning

(b) Q learning

(c) Tabu search

(d) Tabu search

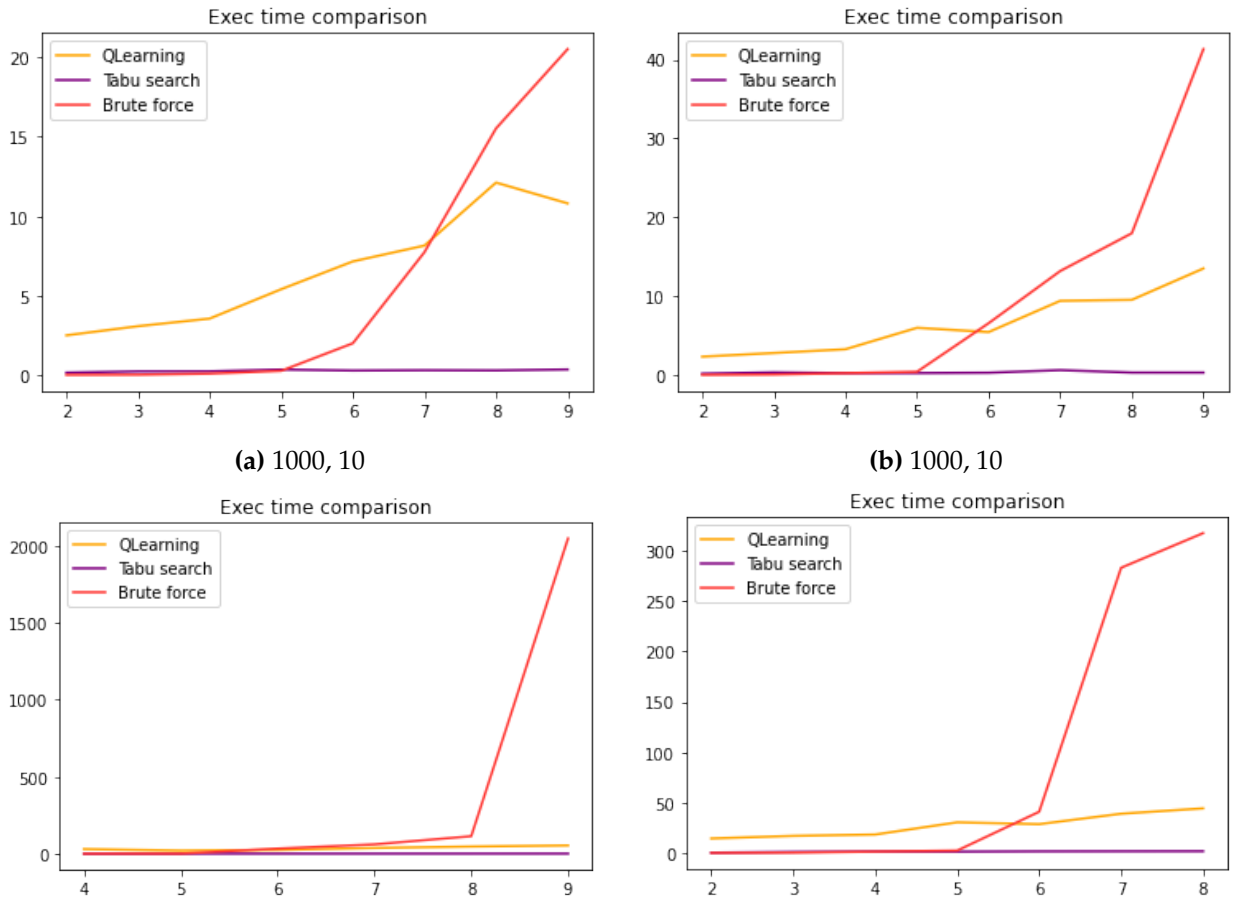
FIGURE 7. – Comparaison du temps d'exécution en fonction du nombre d'enchères

Analyse

Nous voyons clairement une explosion combinatoire à partir de 8-9 enchères pour la méthode force brute. Quant au *Q-learning* il reste relativement constant, même chose pour le *tabu search*.

Autres instances

Afin de conforter les résultats précédents (figure 7) nous effectuons d'autres testes sur différentes instances du problème comme l'illustre la figure 9.



(a) 1000, 10 **(b)** 1000, 10

(c) 6000, 10 **(d)** 5000, 20

FIGURE 8. – Comparaison du temps d'exécution des trois méthodes

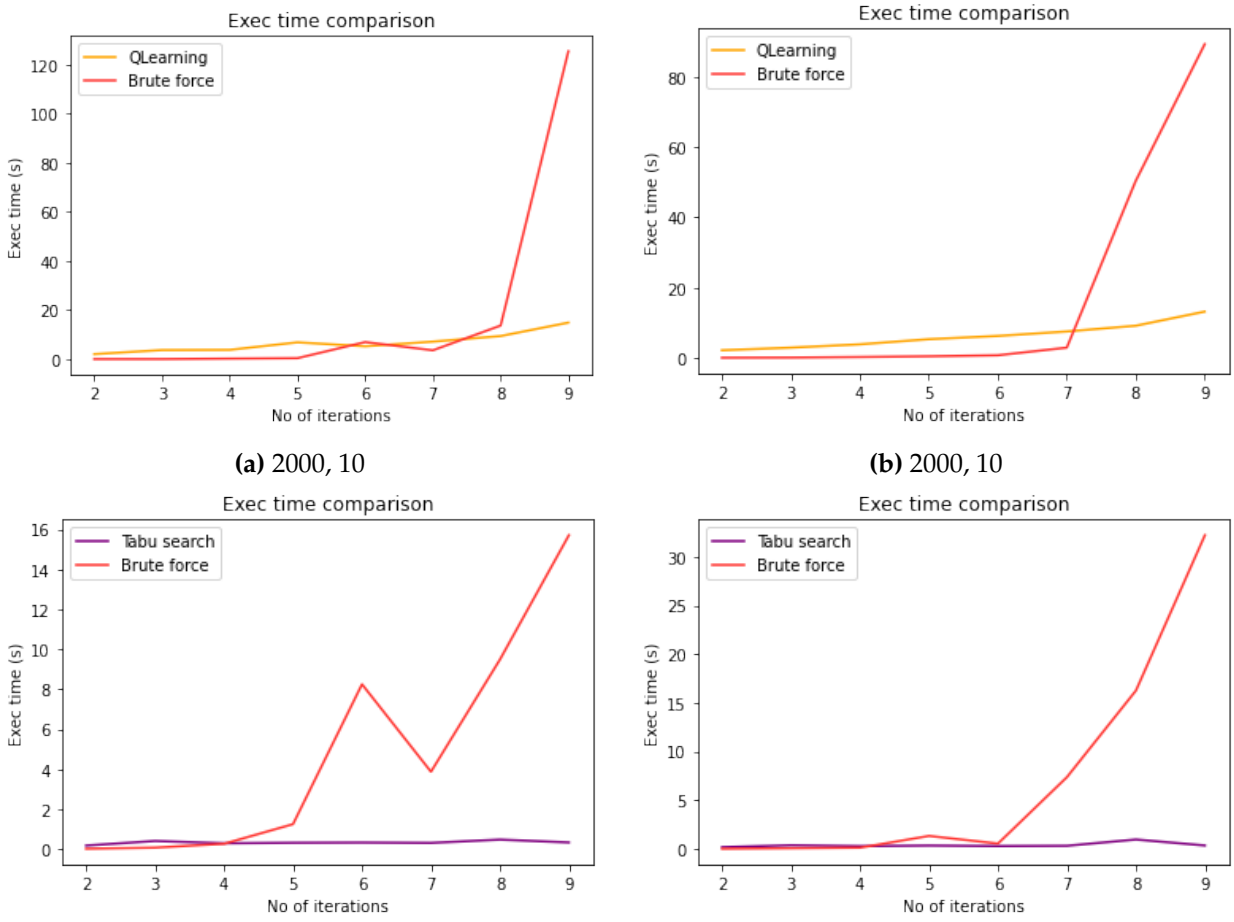
Analyse

Nous voyons que le *tabu search* est plus rapide que le *q learning* et évidement les deux sont nettement plus rapide que le *brute force*.

4 Testes avancés

4.1 Cas triviaux

Instances de 1000 objets, avec 10 objets par enchère. Avec un nombre d'enchère qui varie de 2 à 9.



(a) 2000, 10 **(b)** 2000, 10

(c) 1000, 10 **(d)** 1000, 10

FIGURE 9. – Comparaison du temps d'exécution sur cas triviaux

Analyse

Sur les petites instances (moins de 6 enchères) la *brute force* s'avère très efficace car le nombre total de possibilités est très réduit. Il est même plus rapide que les deux techniques implémentées.

4.2 Cas complexes

Nb enchères	QL		TS	
	Temps	Gain	Temps	Gain
5	3.69	2348	0.99	2348
10	8.49	4598	0.34	4598
15	18.13	3276	0.38	3665
20	42.31	3343	0.41	5332
25	68.77	4145	0.57	4535
29	117.85	4969	0.89	4969

TABLE 2. – Instance de 1000 objets avec 40 à 50 objets par enchère

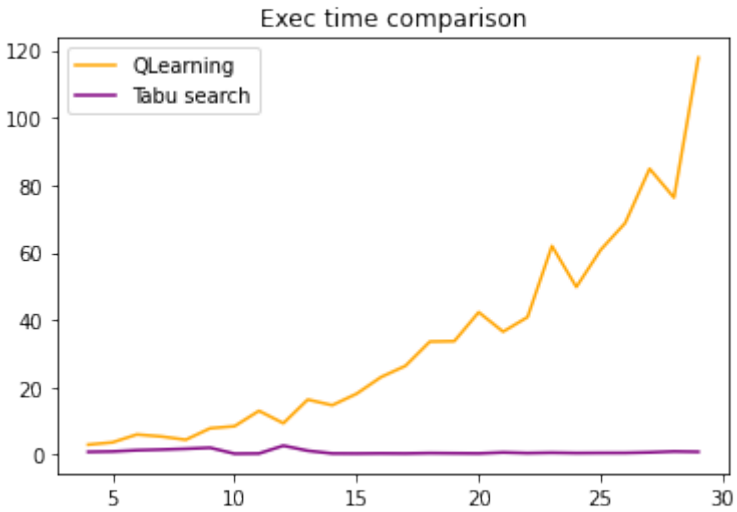


FIGURE 10. – Comparaison du temps d'exécution sur cas complexes

Références

1. FUJISHIMA Y, LEYTON-BROWN K et SHOHAM Y. Taming the Computational Complexity of Combinatorial Auctions : Optimal and Approximate Approaches. In : t. 1. 1999 :548-53.

2. PIEROTTI J, KRONMUELLER M, ALONSO-MORA J, ESSEN JT van et BÖHMER W. Reinforcement Learning for the Knapsack Problem. In : *Optimization and Data Science : Trends and Applications*. Sous la dir. de MASONE A, DAL SASSO V et MORANDI V. Cham : Springer International Publishing, 2021 :3-13.

3. SUR G, RYU SY, KIM J et LIM H. A Deep Reinforcement Learning-Based Scheme for Solving Multiple Knapsack Problems. Applied Sciences 2022 ;12.

4. BENFORD SL. Solving the binary knapsack problem using tabular and deep reinforcement learning algorithms. Northeastern University Boston, Massachusetts, 2021.