

ALGORITHMES POUR MAXIMISER LES BÉNÉFICES

Aide à la décision dans le processus d'achats d'actions

Plan

1 Problème

2 Algorithmes

3 Comparaison des résultats

4 Conclusion

Problème

- ❖ Comment maximiser le profit total des actions achetées, sous les contraintes :
 - Chaque action : **achetée qu'une seule fois.**
 - Acheter une fraction d'action : **non**
 - Dépense maximal : **500 euros**
- => Problème du sac à dos (the 0/1 Knapsack Problem)

Algorithmes

- Les algorithmes pour résoudre le problème :
 - Brute Force
 - Dynamic Programming
 - Greedy
 - Etc.

Représenter le problème

- Étant donnée n actions : A_1, A_2, \dots, A_n
- Noter C_i le coût de A_i
- Noter P_i le profit de la action A_i
=> Le *profit total* de A_i : $C_i * P_i$
(Si P_i représente en « % », le *profit total* de A_i : $C_i * P_i * 0.01$)
- Le problème est de trouver :
(ici, $C_{\max} = 500$)

$$\text{Max}(\sum_i C_i * P_i) \quad \text{sous la contrainte} \quad \sum C_i \leq C_{\max}$$

Algorithme Brute Force

- Lister toutes les différentes combinaisons d'actions
- Pour chaque combinaison, calculer le profit total
- La solution est la combinaison ayant le profit total le plus grand et le coût total ne dépasse pas C_{\max}

Brute Force - Exemple avec 3 actions

- Les combinaisons (lire l'arbre de gauche à droite):

1. A1A2A3

2. A1A2

3. A1 A3

4. A1

5. A2A3

6. A2

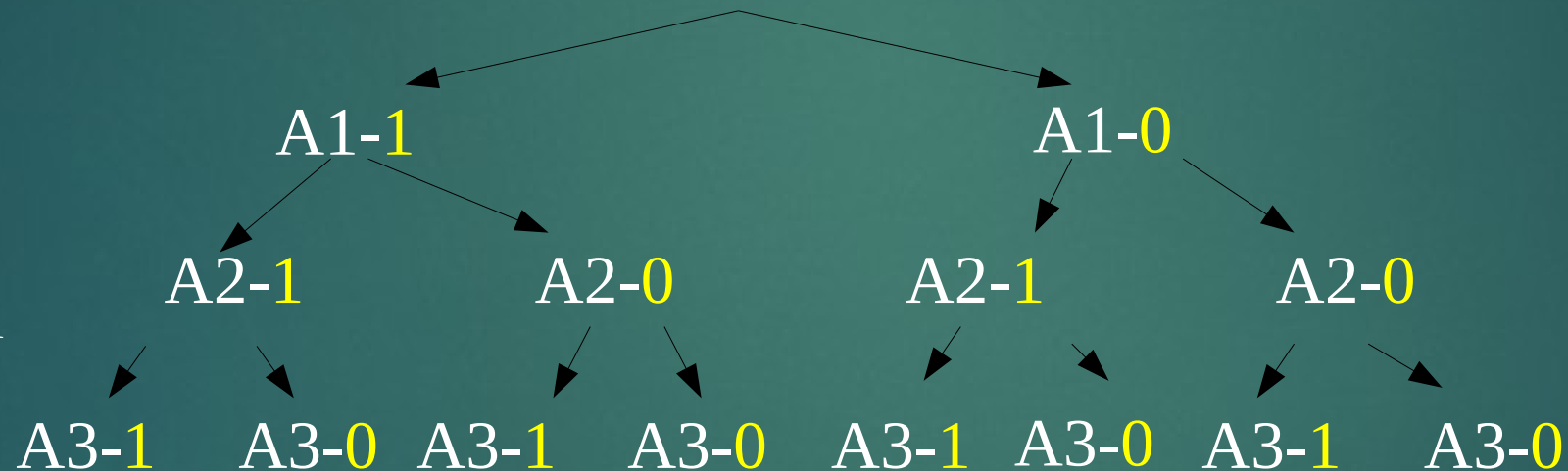
7. A3

8. 0

$\Rightarrow 2^3 = 8$ combinaisons

Chaque combinaison = représentation binaire d'un nombre de 0 à $2^3 - 1$
 $\Leftrightarrow x_1x_2x_3$ où x_1, x_2, x_3 choisi dans $\{0, 1\}$

Exemple en binaire: 111, 110, ..., 011 (=11), ..., 001 (=1), 000 (=0)



Ai-1 : Ai est choisi

Ai-0 : Ai n'est pas choisi

L'arbre représente les choix dans l'algorithme Brute Force

Brute Force - Pseudocode

Complexité

Mémoire / Temporelle
 $O(n)$ $O(n*2^n)$

n = nombre_de_actions

nombre_de_combinaisons = 2^n

solution = []

=> *n éléments*

Pour k de 0 à nombre_de_combinaisons – 1 :

=> *2^n opérateurs*

 choix = représenter k en binaire (en n chiffres binaires)

 combinaison = []

=> *n éléments*

 Pour chaque élément-i dans le choix :

=> *n opérateurs*

 Si élément-i == 1 :

 action-i est choisie donc l'ajouter à la combinaison

Si le coût total de la combinaison $\leq C_{\max}$:

 Si le profit total de la combinaison > le profit total de la solution :

 solution = combinaison

Algorithme Programmation dynamique

- Construire la solution optimale du problème à i actions à partir du problème à $i-1$ actions (les actions sont numérotées $1, 2, \dots, n$)
- Les solutions de sous-problèmes sont pré-calculées et stockées dans un tableau

Trancher le problème avec C_{\max} en des sous problèmes avec le coût max $1, 2, \dots, C_{\max}$

	1	2	...	j	...	C_{\max}
0	0	0	...	0	...	0
A1	$P(\{A1\}, 1)$	$P(\{A1\}, 2)$...	$P(\{A1\}, j)$...	$P(\{A1\}, C_{\max})$
A2	$P(\{A1, A2\}, 1)$	$P(\{A1, A2\}, 1)$...	$P(\{A1, A2\}, j)$...	$P(\{A1, A2\}, C_{\max})$

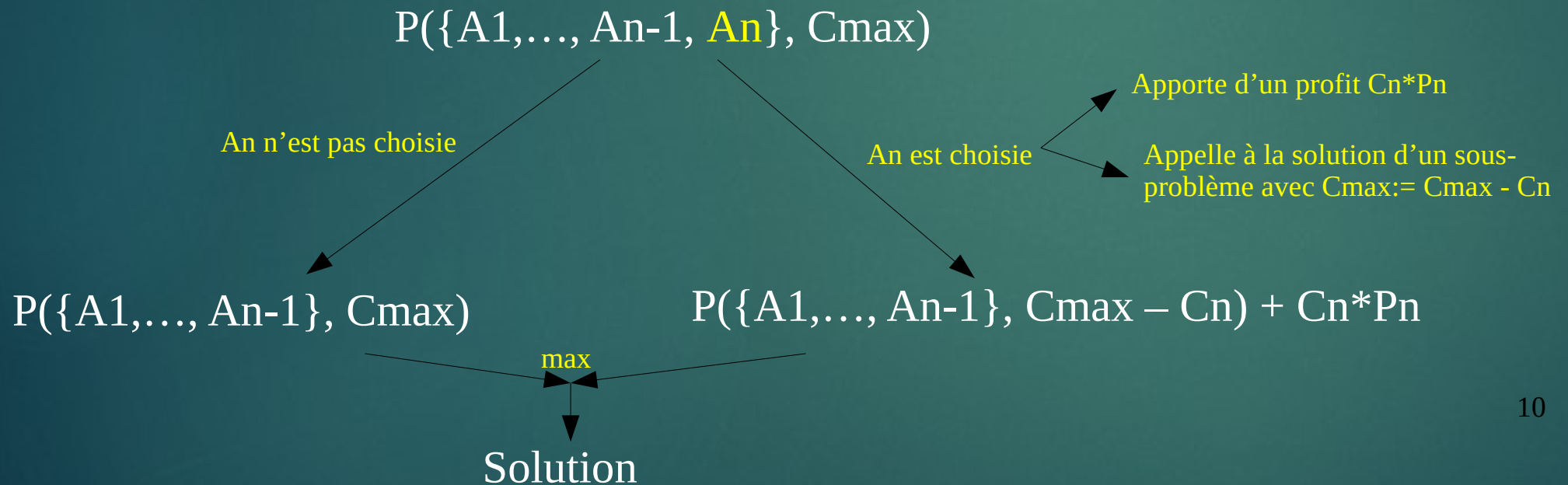
Profit total maximisé

Coût maximal : j

Actions à choisir dans $\{A1, A2\}$

Algorithme Programmation dynamique

	1	C_{\max}
...
A_{n-1}	...	$P(\{A_1, \dots, A_{n-1}\}, C_{\max} - C_n)$...	$P(\{A_1, \dots, A_{n-1}\}, C_{\max})$
A_n	$P(\{A_1, \dots, A_{n-1}, A_n\}, C_{\max})$



Programmation dynamique

Pseudocode

Complexité

Mémoire / Temporelle

$O(n*m)$

$O(n*m)$

$n = \text{nombre_de_actions} + 1$

$m = \text{int}(C_{\max}) + 1$

$B =$ matrice de taille $n*m$ initialisée avec 0

$\Rightarrow n*m$ éléments

pour stocker les profits totales optimales de sous problèmes

remplir la matrice B

Pour i de 1 à n :

 Pour j de 1 à m :

 Si $\text{coût}(\text{action-}i) < j$:

 l'action- i est ajoutée dans la liste des actions à choisir donc

$B[i][j] = \max(B[i-1][j], B[i-1][j - \text{coût}(\text{action-}i)] + \text{profit}(\text{action-}i))$

 Sinon :

$B[i][j] = B[i-1][j]$

$\Rightarrow n*m$

opérateurs

solution = []

$\sim n$ éléments

tracer la solution

Tant que $C_{\max} \geq 0$ et $n \geq 0$:

 Si $B[n][C_{\max}] == B[n-1][C_{\max} - \text{coût}(\text{action-}n)] + \text{profit}(\text{action-}n)$:

 l'action- n est choisie donc l'ajouter dans la solution

$C_{\max} = C_{\max} - \text{coût}(\text{action-}n)$

$n = n - 1$

$\sim n*m$ opérateurs

Résumé

n : nombre d'actions

m : coût total maximal autorisé

	Complexité en mémoire	Complexité temporelle	Remarque
Brute Force	$O(n)$	$O(n \cdot 2^n)$	Temps de calcul exponentiel
Programmation dynamique	$O(n \cdot m)$	$O(n \cdot m)$	<ul style="list-style-type: none">- Temps de calcul important si m est grand- Coût total autorisé et le coût de chaque action doivent être un nombre entier