Plan

- Introduction
- Programmation dynamique
  - Partition
  - Sac à dos
  - Voyageur de commerce
- Branch & bound
- Comparaison
- Conclusion

INTRODUCTION

Programmation dynamique

- Paradigme d'algorithmes exacts
- Plongement du problème dans sa généralisation
- Fonction de récurrence

Branch & Bound

- Méthode exacte
- Arbre représentant l'espace des solutions
- Méthode de séparation
- Méthode d'évaluation

- Partition

Programmation dynamique

Problème de la partition

- Problème de décision
- Construction de 2 sous-ensembles E_1, E_2
- $|E_1| = |E_2|$

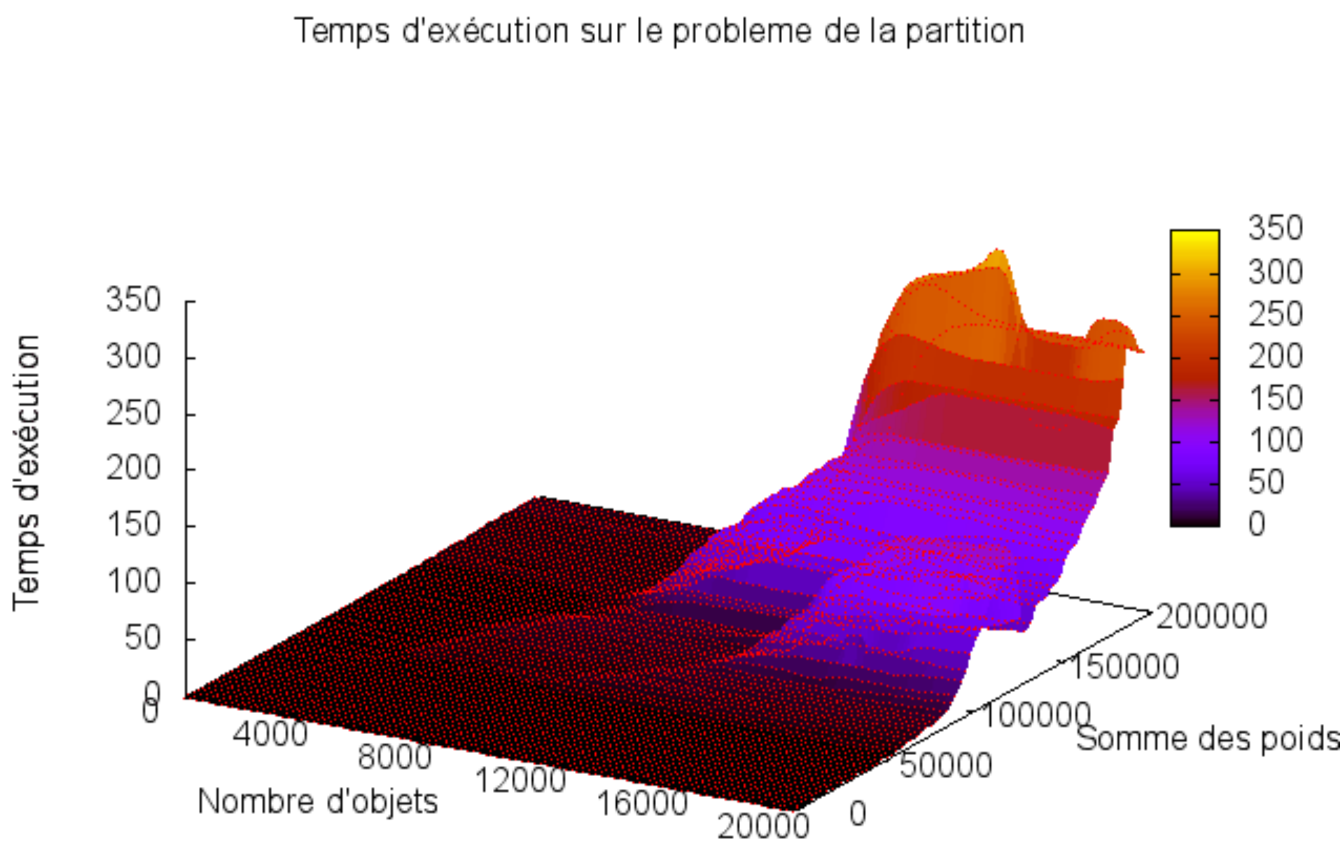
Algorithme

- si le poids total est impair, renvoyer faux
- création de T : (n, P/2)-matrice de bool
- initialisation
  - $T[0,0]$ = vrai ; pour $j \in [0,P/2]$: $T[0,j]$ = faux
- pour $i \in [1,n]$
  - pour $j \in [0,P/2]$
    - $T(i,j) \leftarrow [(j = 0) \vee (j = p(a_i)) \vee (T(i-1, j)) \vee (T(i-1, j-p(a_i)))]$
- renvoyer $T[n, P/2]$

Implémentation

- Langage C
- Complexité
  - Temps : O(n.P)
  - Espace : O(n.P)

Tests



Temps d'exécution sur le probleme de la partition
Temps d'exécution
350
300
250
200
150
100
50
0
4000
8000
12000
16000
20000
Nombre d'objets
0
50000
100000
150000
200000
Somme des poids

- Sac à dos

Programmation dynamique

Problème du sac à dos

- Problème d'optimisation
- Remplir un sac à dos avec *n* objets
- Volume limité
- Choix d'une solution à utilité maximale

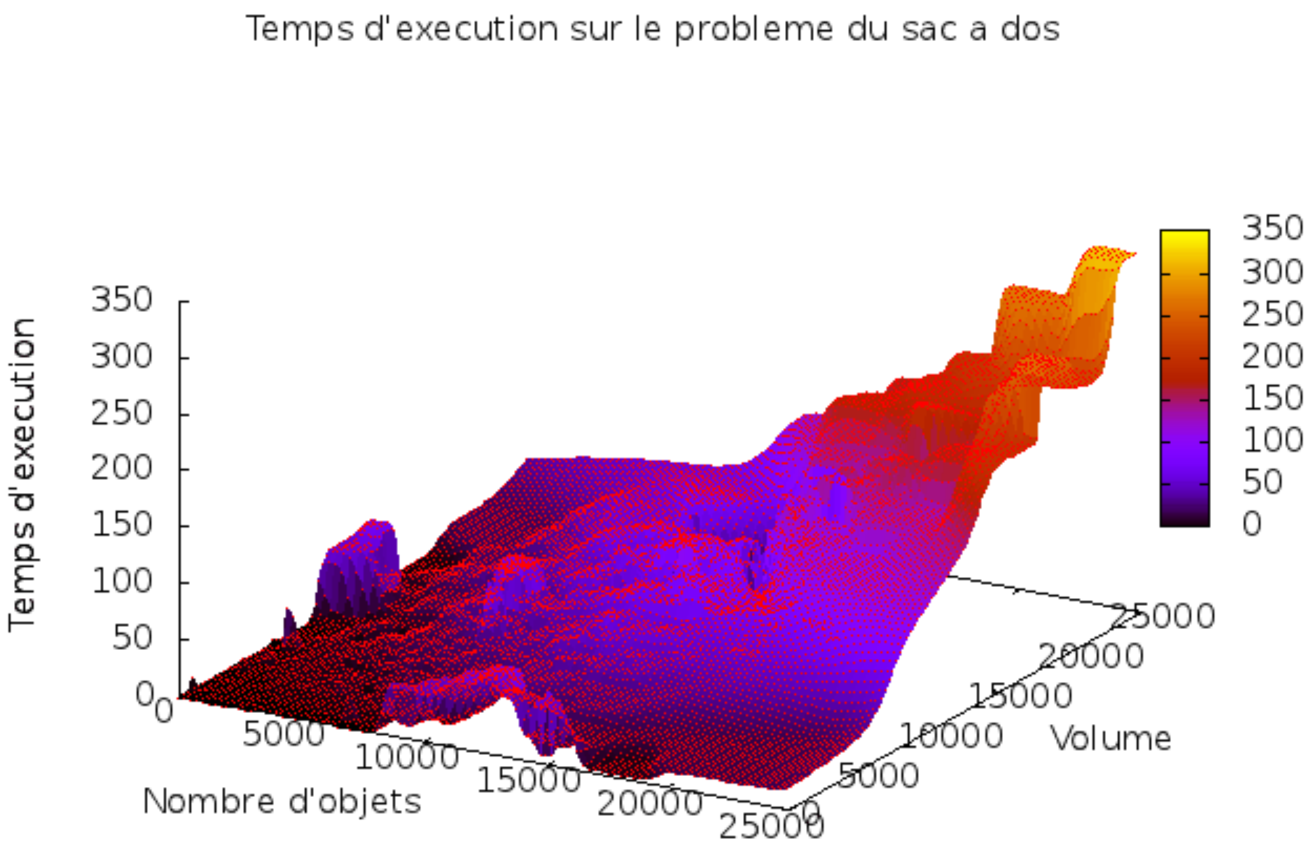
Algorithme

- création de T une (n+1, volumeMax)-matrice
- initialisation : pour $j \in \{1, \ldots, volumeMax\}$
  - $T[0,j] = 0$
- pour $i \in \{1,\ldots,n\}$
  - pour $j \in \{1, \ldots, volumeMax\}$
    - pour $k \in \{0, \ldots, volumeMax/volume[i]\}$
      - $T[i, j] \leftarrow \max(T[i, j], T[i-1, j-k \times volume[i]] + k \times utilite[i])$
- renvoyer T[n,volumeMax]

Implémentation

- Langage C
- Complexité
  - Temps : $O(n.V^2)$
  - Espace : $O(V)$

Tests


Temps d'execution sur le probleme du sac a dos
Temps d'execution
350
300
250
200
150
100
50
0
Nombre d'objets
0
5000
10000
15000
20000
25000
Volume
0
5000
10000
15000
20000
25000
350
300
250
200
150
100
50
0

- Voyageur de commerce

Programmation dynamique

Problème du voyageur de commerce

- Passer une seule fois par chaque ville
- Revenir au point de départ
- Voyage de coût minimum

- cycle hamiltonien de poids minimum

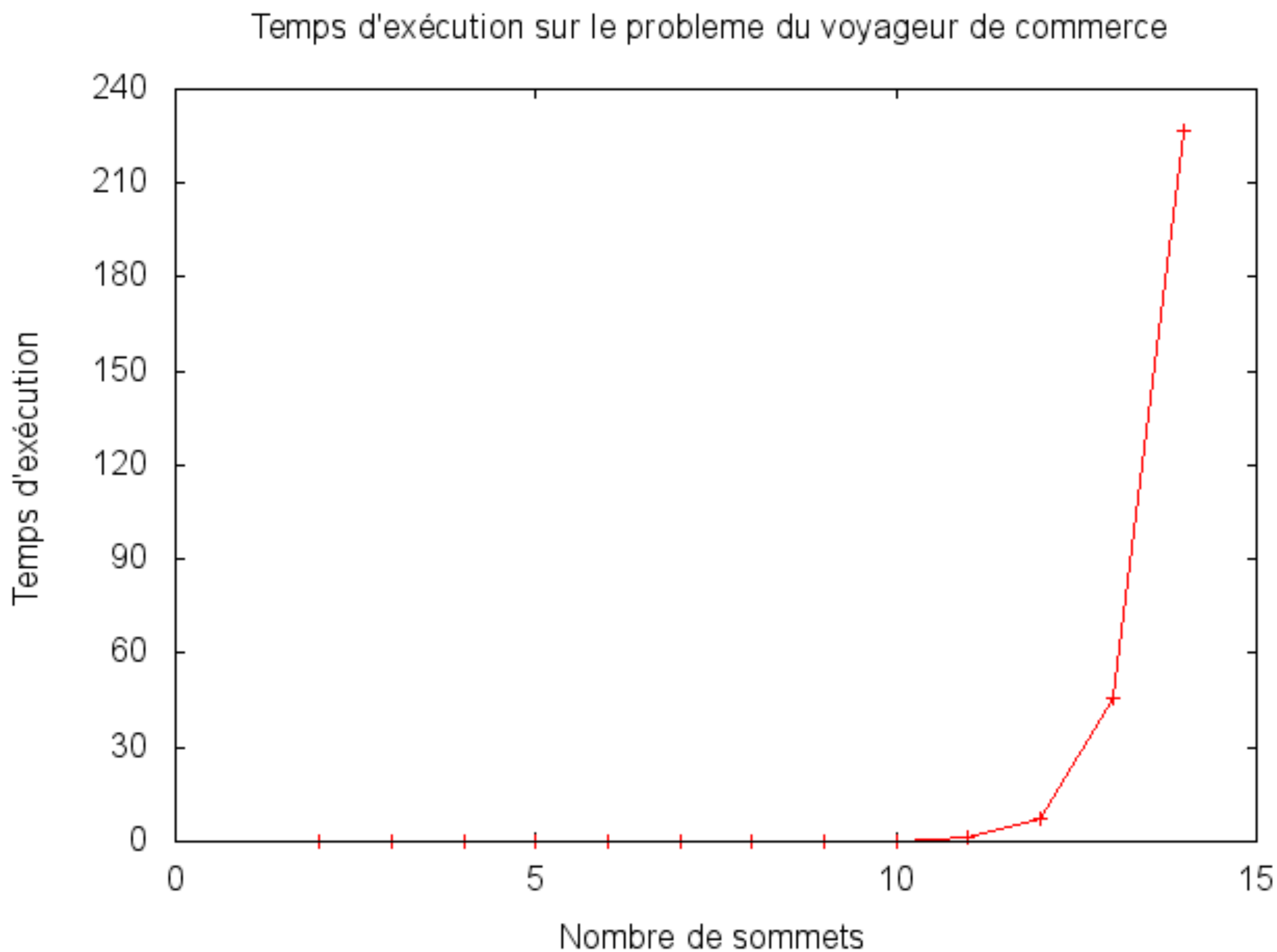
Algorithme

- Initialisation : pour $i \in \{1, \ldots, n\}$
  - $C(\{0\},i) = p(0,i)$
- pour $i \in \{2, \ldots, n\}$
  - pour $S \subseteq \{1, \ldots, n\} : |S|=i$
    - pour $j \in V \backslash S$
      - $C(S,j) = \min_{k \in S \backslash \{0\}}(C(S \backslash \{k\}, k) + p(k, j))$
- Renvoyer $\min_{i \in V \backslash \{0\}}(C(V \backslash \{i\},i) + p(i, 0))$

Implémentation

- Langage C
- Complexité
  - Temps : $O(n^2.2^n)$
  - Espace : $O(2^n)$

Tests


Temps d'exécution sur le probleme du voyageur de commerce
Temps d'exécution
240
210
180
150
120
90
60
30
0
0
5
10
15
Nombre de sommets

BRANCH & BOUND

Solutions initiales admissibles au TSP

- Chaine de poids le plus faible
- Voisinage 2-opt
- Voisinage 3-opt
- Solution 3/2 approchée

Comparaison des résultats

COMPARAISON P.D. VS B&B POUR LE TSP

Résultats

CONCLUSION

Prog. Dyn. vs B&B

- Programmation dynamique
  - Paradigme intuitif
  - Occupation mémoire importante
  - Efficace sur des petites valeurs
- Branch&Bound
  - Difficulté à déterminer les fonctions
  - Efficace sur les grandes valeurs

Merci de votre attention !
Avez-vous des questions ?