Rapport du projet de programmation linéaire

Maxence Ahlouche Maxime Arthaud Korantin Auguste
Martin Carton Thomas Forgione Thomas Wagner

?

Table des matières

1 Problème du sac à dos

Nous avons implémenté un algorithme de programmation dynamique, qui permet de résoudre le problème du sac à dos. Toutefois, il fonctionne uniquement si les poids des objets sont des entiers.

Sa complexité en temps et en mémoire est en $O(nW)$, avec n le nombre d'objets et W le poids maximum du sac.

Nous l'avons testé sur plusieurs instances du problème (jusqu'à X objets et un poids maximal de X), et l'algorithme s'exécute toujours en moins d'une seconde.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import numpy

def sacados(objets, masse_max):
    """
    Résoud le problème du sac à dos avec de la programmation dynamique.
    Fonctionne seulement avec des valeurs entières.
    On pourrait optimiser l'algorithme en ne retenant que la ligne pour (i-1), et
        pas toute les lignes.

    >>> objets = ((2,3),(3,4),(4,5),(5,6))
    >>> sacados(objets, 5)
    7
    >>> objets =
        ((5,3),(1,1),(1,3),(4,4),(4,1),(3,5),(1,1),(5,2),(5,2),(1,3),(2,1),(2,4))
    >>> sacados(objets, 6)
    12
    """
    assert isinstance(masse_max, int) and all(lambda x: isinstance(x[0], int) for
        x in objets)

    matrice = numpy.zeros(shape=(len(objets)+1, masse_max+1), dtype='int64')

    for i in range(1,len(objets)+1):
        masse_objet, prix = objets[i-1]
        for masse in range(masse_max + 1):
            if masse_objet <= masse:
                matrice[i,masse] = max(matrice[i-1,masse],
                    matrice[i-1,masse-masse_objet] + prix)
            else:
                matrice[i,masse] = matrice[i-1,masse]

    return matrice[len(objets)-1,masse_max]


if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Nous aurions aussi pu faire un algorithme glouton, en triant les objets par rapport prix / poids.