Rapport du projet de programmation linéaire

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

21 octobre 2013

Table des matières

1 Présentation de l'équipe

Cette équipe a été menée par Maxence Ahlouche, assisté de son Responsable Qualité Thomas Wagner. Les autres membres de l'équipe sont Martin Carton, Thomas Forgione, Maxime Arthaud, et Korantin Auguste.

2 Problème du sac à dos

2.1 Résolution exacte

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

Nous l'avons testé [1] 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.

2.2 Résolution approchée

Nous avons aussi implémenté l'algorithme glouton : celui-ci consiste à choisir les « meilleurs » objets jusqu'à que la masse maximale soit dépassée. Le critère déterminant quels sont les meilleurs objets peuvent être la masse faible, le prix élevé, ou le rapport prix/masse élevé.

Cet algorithme est beaucoup plus rapide que le précédent, mais n'est qu'un algorithme approché. La table 1 montre quelques-uns des résultats obtenus.

On remarque qu'en général, la résolution approchée en considérant le ratio prix/masse donne de très bon résultats, voire le meilleur résultat. Le résultat que fournit cet algorithme est le moins bon quand la masse maximale autorisée est faible comparée à l'amplitude des valeurs que peuvent prendre le prix et la masse des objets.

1. En utilisant le générateur de problèmes trouvé à l'adresse suivante : `http://www.diku.dk/~pisinger/codes.html`.

Paramètres du générateur/ masse maximale autorisée	Résultat optimum	Prix le plus élevé	Masse la plus faible	Meilleur ratio prix/masse
50 25 1 1 1000/20	85	49/42.4%	67/21.2%	81/4.7%
500 25 1 1 1000/500	2016	1125/44.2%	1725/14.4%	1983/1.6%
5000 25 1 1 1000/500	5540	1175/79%	4577/17.4%	5540/0%
50000 25 1 1 1000/500	11195	1175/90%	6684/40.3%	11195/0%
50000 1000 1 1 1000/500	118260	5959/95%	101857/13.9%	118147/0.1%
50000 5000 1 1 1000/100	100847	14931/85.2%	93532/7.3%	100282/0.6%

TABLE 1 – Résultats de l'algorithme glouton

3 Problème d'optimisation

Le but est maximiser une fonction linéaire sous certaines contraintes (inéquations linéaires).

3.1 Simplexe

L'algorithme du simplexe a une complexité dans le pire des cas exponentielles, mais en pratique, cet algorithme est efficace.

expliquer principes algo

4 Annexe

Listings

Listing 1 – Codes relatifs au problème du sac à dos

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

def sacados(objets, masse_max):
    """
    Résoud le problème du sac à dos avec de la programmation dynamique.
    Fonctionne seulement avec des valeurs entières.

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

    current_line = [0 for i in range(masse_max+1)]
    prev_line = current_line[:]

    for i in range(0,len(objets)):
```

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

        prev_line = current_line[:]

    return current_line[masse_max]

def best_ratio(x): return x[1]/x[0]
def less_mass(x):  return -x[0]
def best_price(x): return x[1]

def greedy(objects, max_mass, key):
    """
        Algorithme approché du glouton.
        Nécessite de trier les objets selon un critère `key`.
        Par exemple
            greedy(obj, max_mass, less_mass)
        choisit les objets en commençant par les moins lourds.
    """
    cost, mass = 0, 0
    objects = sorted(objects, key=key, reverse=True)

    for o in objects:
        if o[0] + mass <= max_mass:
            mass += o[0]
            cost += o[1]

            if mass == max_mass:
                break

    return cost

def read_testfile(path):
    """
        Lit un fichier généré par le générateur trouvé ici:
            http://www.diku.dk/~pisinger/codes.html
        Retourne une liste de couples (masse, valeur) considérée comme bon
        exemple.
    """
    with open(path, 'r') as f:
        objects = []
        line = f.readline()
        nb_objs = int(line)
        for i in range(0, nb_objs):
            line = f.readline()
            dummy, a, b = map(int, line.split())
            objects.append((b, a))
        return objects

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

Listing 2 – Codes relatifs au simplexe

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
```

```python
from __future__ import division
import numpy as np

"""
La matrice d'entrée doit avoir la forme suivante :

À chaque colonne correspond un produit :
[benef_du_produit, cout_ressource_1, cout_ressource_2, ...]
(c'est les variables libres)

Sur les dernières colonnes, on a les variables de base :
[0, 0, ..., 0, 1, 0, ..., 0]
sachant que le carré des variables de base forme une matrice identité.

Et sur la toute dernière colonne :
[0, stock_ressource_1, stock_ressource_2, ...]

"""

def simplexe(matrice):
    size_y, size_x = matrice.shape

    # indice de colonne (pour le x à virer de la base)
    a_virer = np.argmin(matrice[0,])
    if matrice[0,a_virer] >= 0:
        return matrice[0,-1]

    # indice de ligne (pour le x à ajouter dans la base)
    a_ajouter = None
    meilleur_ratio = 0
    for y in range(1, size_y): # la première ligne est pour z, osef
        if matrice[y,a_virer] == 0:
            continue
        ratio = matrice[y,-1] / matrice[y,a_virer]
        if a_ajouter is None or ratio < meilleur_ratio:
            a_ajouter = y
            meilleur_ratio = ratio

    # opérations sur les lignes
    for y in range(size_y):
        if y == a_ajouter:
            matrice[y,] /= matrice[y,a_virer]
        else:
            ratio = matrice[y,a_virer] / matrice[a_ajouter, a_virer]
            matrice[y,] -= ratio * matrice[a_ajouter,]

    return simplexe(matrice)
```