Numérique et science informatique
Classe de première

Lycée hoche

année scolaire 2025-2026

Contents

1 Algorithmes gloutons

Optimiser un problème, c'est déterminer les conditions dans lesquelles ce problème présente une caractéristique spécifique.Par exemple, déterminer le minimum ou le maximum d'une fonction est un problème d'optimisation.

On peut également citer la répartition optimale de tâches suivant des critères précis, le **problème du rendu de monnaie** , le problème du sac à dos , la recherche d'un plus court chemin dans un graphe , le problème du voyageur du commerce.

De nombreuse techniques informatiques sont susceptibles d'apporter une solution exacte ou approchée à ces problèmes.
Certaines de ces techniques , comme l'énumération exhaustive de toutes les solutions , ont un coût machine qui les rend souvent peu pertinentes au regard de contraintes extérieures imposées (temps de réponse de la solution imposé, moyens machines limités).

Les techniques de *programmation dynamique* ou *d'optimisation linéaire*, certains algorithmes numériques peuvent apporter une solution.

Les **algorithmes gloutons** constituent une alternative dont le résultat n'est pas toujours optimal.

Plus précisément , **ces algorithmes déterminent une solution optimale en effectuant successivement des choix *locaux* , jamais remis en cause.**

Au cours de la construction de la solution, l'algorithme résout une partie du problème puis se focalise ensuite sur le sous-problème restant à résoudre.

Le principal avantage des algorithmes gloutons est leur facilité de mise en oeuvre.En outre , dans certaines situations dites *canoniques*, il arrive qu'ils renvoient non pas un optimum mais l'optimum du problème.Nous présentons de telles situations dans la suite de ce chapitre, en montrant les avantages mais aussi les limites de cette technique.

2 Rendu de monnaie

Présentation du problème

Un achat en espèces se traduit par un échange de pièces et de billets.Dans la suite de cet exposé, les pièces désignent indifféremment les véritables pièces que les billets.

Supposons qu'un achat induise un rendu de 49 euros.Quelles pièces peuvent être rendues?
La réponse , bien qu'évidente, n'est pas unique:
Quatre pièces de 10 euros, 1 pièce de 5 euros et deux pièces de 2 euros conviennent.Mais quarante neufs pièces de 1 euro conviennent également.

Si la question est de rendre la monnaie avec un minimum de pièces , le problème change de nature.Mais la réponse reste simple: C'est la première solution proposée.
Toutefois,comment parvient-on à un tel résultat?Quels choix ont été faits qui optimisent le nombre de pièces rendues?
C'est le problème du rendu de monnaie dont la solution dépend du système de monnaie utilisé.

Dans le système de monnaie français (ou de n'importe quel pays ayant pour monnaie l'euro) , les pièces ou billets prennent les valeurs 1,2,5,10,20,100 euros.

Pour simplifier, nous nous intéresserons seulement aux valeurs entières et oublions l'existence du billet de 500 euros.

Rendre 49 euros avec un minimum de pièces est un problème d'optimisation.

En pratique, sans s'en rendre compte généralement , tout individu met en oeuvre un algorithme glouton:
Il choisit d'abord la plus grande valeur de monnaie , parmi 1,2,5,10,20,100 contenue dans 49 euros.En

l'occurence 2 pièces de 20 euros .La somme de 9 euros restant à rendre , il choisit une pièce de 5 euros , puis deux pièces de 2 euros.

Cette stratégie gagnante pour la somme de 49 euros l'est-elle pour n'importe quelle somme à rendre?On peut montrer que la réponse est positive pour le **système monétaire français**.Pour cette raison , un tel système de monnaie est **qualifié de canonique**.

D'autres systèmes ne sont pas canoniques.L'algorithme glouton ne répond alors pas de manière optimale.

Par exemple , avec le système $\{1, 3, 6, 12, 24, 30\}$, l'algorithme répond en proposant
le rendu $49 = 30 + 12 + 6 + 1$, soit 4 pièces alors que la solution optimale est $49 = 2 \times 24 + 1$ soit 3 pièces.

2.1 Un algorithme glouton

Considérons un ensemble de n pièces de monnaie de valeurs:

$$v_1 < v_2 < ... < v_n$$

avec $v_1 = 1$ .On suppose que ce système est canonique.On peut noter le système de pièces :

$$S_n = \{v_1, ..., v_{n-1}, v_n\}$$

Désignons par s une somme à rendre avec le minimum de pièces de S_n.L'algorithme glouton sélectionne la plus grande valeur v_n et la compare à s.

- Si $s < v_n$, la pièce de valeur v_n ne peut pas être utilisée.

  On reprend l'algorithme avec le système S_{n-1}.

- Si $s \geqslant v_n$, la pièce v_n peut être utilisée une première fois.Ce qui fait une première pièce à comptabiliser , de valeur v_n , la somme restant à rendre étant alors $s - v_n$.

  L'algorithme continue avec le même système de pièces S_n et cette nouvelle somme à rendre $s - v_n$.

L'algorithme est ainsi répété jusqu'à obtenir une somme à rendre nulle.

Remarque : Il s'agit effectivement d'un algorithme glouton , la plus grande valeur de la pièce étant systématiquement choisie si sa valeur est inférieure à la somme à rendre.Ce choix ne garantit en rien l'optimalité globale de la solution.
Le choix fait est considéré comme pertinent et permet d'avancer plus avant dans le calcul.Toutefois, comme dit plus haut , **si le système monétaire est canonique , la solution est optimale.**

2.2 Code

On utilise : systeme_monnaie = [1,2,5,10,20,50,100]

Le code précédent peut être écrit en utilisant une fonction qui reçoit deux arguments - la somme à rendre et le système de monnaie - et qui renvoie la liste des pièces choisies par l'algorithme glouton.

Ecrire le pseudo-code qui traduit l'algorithme ci-dessus , puis le programmer en langage Python

3 Un problème d'organisation

Des conférenciers sont invités à présenter leurs exposés dans une salle,mais leurs disponibilités ne leur permettent d'intervenir qu'à des horaires bien définis.

Le problème est de construire un planning d'occupation de la salle avec *le plus grand nombre* de conférenciers.

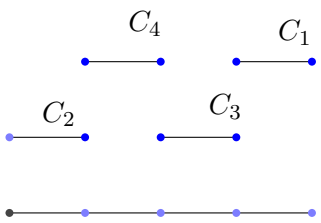
Désignons par n , entier naturel , le nombre de conférenciers.Chacun d'eux, identifié par une lettre C_i , où i est un entier compris entre 0 et $n-1$, est associé à un intervalle temporel $[d_i, f_i]$ où d_i et f_i désignent respectivement l'heure de début et l'heure de fin de l'intervention.Afin de dégager une tactique de résolution du problème , commençons par analyser plusieurs situations.

3.1 Premières analyses

situation 1

Quatre conférenciers peuvent intervenir aux intervalles temporels suivants , illustrés sur la figure :

$C_1 : [3;4[$, $C_2 : [0;1[$ $C_3 : [2;3[$, $C_4 : [1;2[$

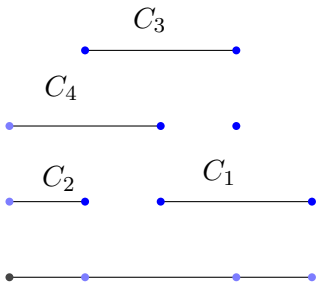


Une telle situation est simple puisque tous les conférenciers peuvent intervenir sur des créneaux horaires disjoints.Le planning est donc défini par la suite $[C_2, C_4, C_3, C_1]$ de conférenciers.
L'algorithme menant à ce résultat choisit les conférenciers par ordre croissant des heures de début ou de fin des conférences après s'être assuré que les intervalles sont disjoints.

Situation 2

On considère à nouveau quatre conférenciers dont les créneaux horaires ne sont plus toujours disjoints.

$C_1 : [2;4[$, $C_2 : [0;1[$ $C_3 : [1;3[$, $C_4 : [0;2[$



Dans cette situation, les intervalles **ne sont plus disjoints**.Nous dirons que **ces intervalles ne sont pas compatibles**.

Des choix doivent êtres faits et certains conférenciers peuvent ne pas être retenus pour construire un planning.Plusieurs solutions peuvent être construites:

$[C_2, C_1]$ ou $[C_2, C_3]$ ou $[C_4, C_1]$ ou $[C_3]$

Seules les trois premières solutions sont retenues puisque la dernière ne maximise pas le nombre de conférenciers choisis.Mais laquelle de ces solutions retenir?

Pour y répondre, il convient de se demander comment un algorithme pourrait aboutir à la construction de ces solutions.
Une idée serait de nouveau de classer par ordre croissant les heures de début des intervalles compatibles.

En procédant de la sorte, C_1 et C_2 sont compatibles avec $d_2 < d_1$; ce qui mène au planning $[C_2, C_1]$.De même C_2 et C_3 sont compatibles avec $d_2 < d_3$ ce qui mène à $[C_2, C_3]$.Enfin, C_4 et C_1 sont compatibles avec $d_4 < d_1$ ce qui mène à $[C_4, C_1]$.

Une autre idée serait de construire les plannings en classant par ordre croissant les heures de fin des intervalles compatibles.En procédant de la sorte, on retrouve les trois propositions de plannings précédentes.

Il semble donc que ces deux idées mènent à des résultats identiques.En outre, elles n'ont pas permis d'éliminer des solutions afin de n'en fournir qu'une seule.

C'est à ce niveau que la stratégie gloutonne intervient.

Celle-ci va faire un premier choix de conférencier en suivant un critère à préciser.Ce choix ne sera jamais remis en question et la même stratégie sera appliquée pour trouver les conférenciers suivants.

Après avoir classé les intervalles par valeurs croissantes des heures de début , sélectionner l'intervalle de la première plus petite valeur, puis celui de la deuxième plus petite valeur compatible avec la précédente, et ainsi de suite.On observe que:

$$d_2 = d_4 < d_3 < d_1$$

et que
C_2 et C_4 ne sont pas compatibles.
C_3 et C_4 ne sont pas compatibles.
C_1 et C_3 ne sont pas compatibles.

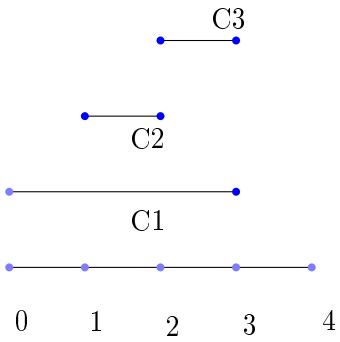
Deux solutions $[C_2, C_3]$ et $[C_4, C_1]$ restent , en raison de l'égalité $d_2 = d4$ qui ne permet pas de choisir le premier conférencier.On peut aussi classer les intervalles par valeurs croissantes des heures de fin , puis sélectionner l'intervalle de la première plus petite valeur,puis celui de la deuxième plus petite valeur compatible avec la précédente et ainsi de suite.On observe que:

$$f_2 < f_4 < f_3 < f_1$$

avec les mêmes incompatibilités que précédemment.Une seule solution est alors possible : $[C_2, C_3]$. Cette solution était également proposée par la stratégie précédente.Il est alors légitime de se demander si ce dernier résultat relève d'une stratégie générale pertinente ou d'une situation trop particulière.La situation suivante apporte un premier élément de réponse.

3.2 Situation 3

Considérons à présent trois conférenciers



Appliquons les deux stratégies précédentes:

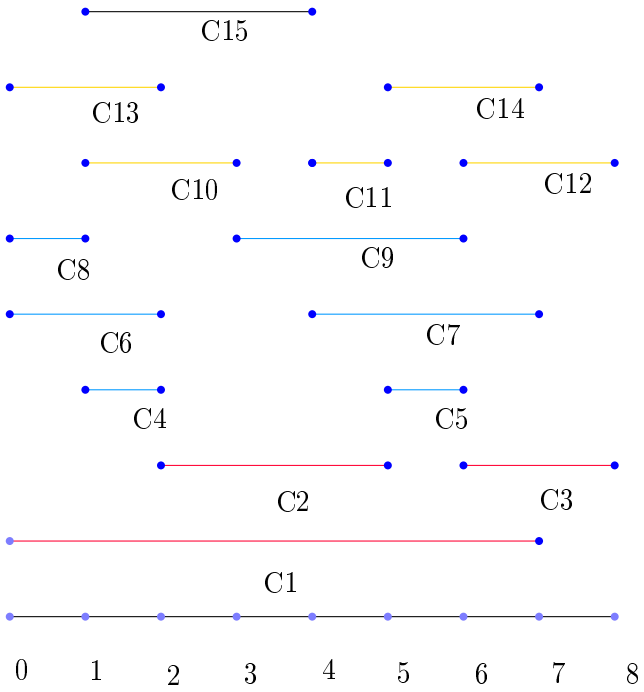
1. On classe les heures de début : $d_1 < d_2 < d_3$.
   Seuls C_2 et C_3 sont compatibles.Mais puisque d_1 est la plus petite des heures, le planning proposé en suivant cette stratégie se réduit à $[C_1]$ alors que $[C_2, C_3]$ est une meilleure solution puisqu'elle maximise le nombre de conférenciers.
2. En classant les heures de fin , on a $f_2 < f_1 = f_3$.Cette fois-ci le planning proposé en suivant la seconde stratégie fournit le planning $[C_2, C_3]$.

3.3 Présentation de l'algorithme glouton pour ce problème d'organisation

Une solution semble émerger des observations précédentes.On peut proposer la stratégie suivante:

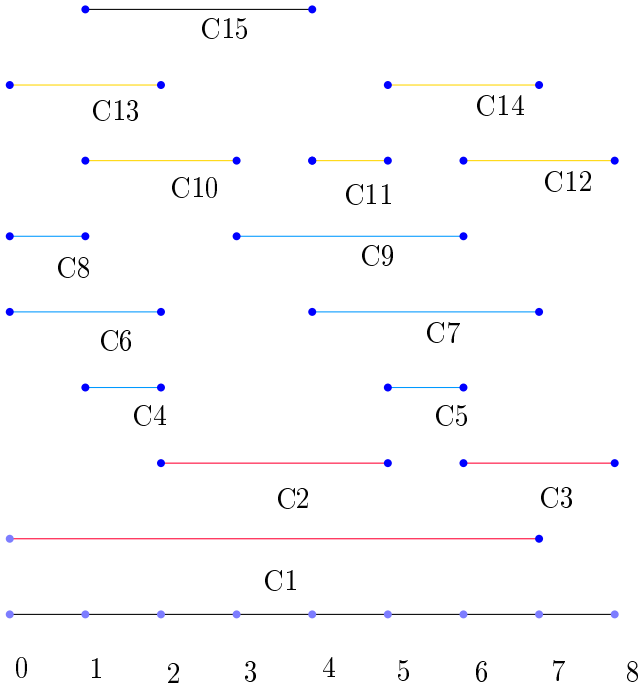
1. Classer les intervalles par heures de fin croissantes.
2. Choisir le conférencier associé au premier intervalle.
3. Choisir parmi les intervalles suivants celui du conférencier dont l'intervalle est compatible avec celui du conférencier précédemment choisi.
4. Recommencer ainsi avec les intervalles classés suivants jusqu'à ce qu'il n'y en ait plus à traiter.

Une situation plus complexe



Commençons par classer les conférenciers par heures de fin croissantes en notant $\preccurlyeq$ la relation d'ordre associée.

$C_8 \preccurlyeq C_4 \preccurlyeq C_6 \preccurlyeq C_{13} \preccurlyeq C_{10} \preccurlyeq C_{15} \preccurlyeq C_2 \preccurlyeq C_{11} \preccurlyeq C_5 \preccurlyeq C_9 \preccurlyeq C_1 \preccurlyeq C_7 \preccurlyeq C_{14} \preccurlyeq C_3 \preccurlyeq C_{12}$



Solution :

On construit petit à petit le planning.

- Le premier conférencier est C_8 .
- Le conférencier suivant dont l'intervalle est compatible avec celui de C_8 est C_4.
- Le conférencier suivant compatible avec C_4 est C_2.
- Le conférencier suivant compatible avec C_2 est C_5.

- Le conférencier compatible avec C_5 est C_3.

D'où la solution $[C_8, C_4, C_2, C_5, C_3]$

Remarque 1 : Cet algorithme est effectivement glouton.Chaque choix fait sélectionne l'une des meilleures possibilités et ne remet pas en cause les choix précédents.

Remarque 2 : Pour conclure cette proposition d'algorithme , nous admettons que cette solution est optimale , c'est-à-dire de cardinal maximal.

```
from random import *
import matplotlib.pyplot as plt

debut=0
fin =8
duree_max=5
NOMBRE_INTERVALLES = 10
def intervalles(nb_intervalles):
    tab_intervalles=[]
    for i in range(nb_intervalles):
        duree=randint(1,duree_max)
        deb=randint(debut,fin - duree)
        tab_intervalles.append([deb,deb+duree,'C'+str(i)])
    print(tab_intervalles)
    return tab_intervalles
```

```
def planning(tab_intervalles):
    nb_intervalles=len(tab_intervalles)
    #tri des intervalles par valeurs croissantes des heures de fin
    tab_intervalles = ..................................
    #tableau du planning
    tab_planning=..............................
    j=0
    #on parcourt les intervalles de la liste ordonnée des intervalles
    #on regarde si l'heure de début de tab_intervalles[i] est supérieure
    #à l'heure de fin de tab_intervalles[j] en application
    #de l'algorithme glouton
    for i in range(1,nb_intervalles):
        if.....................................
      #si les deux intervalles sont compatibles ,on accepte l'intervalle
       # tab_intervalle[i]
            .....................................
           j=.................
    return tab_planning
```

```
def affichage():
    conferenciers=planning(intervalles(NOMBRE_INTERVALLES))

    #l'affichage
    for i in range(len(conferenciers)):
        Y=[i+1,i+1]
        a=(conferenciers[i][0]+conferenciers[i][1])/2
        texte=conferenciers[i][2]
        plt.plot(conferenciers[i][:2],Y, linestyle = 'solid',\
        marker = 'o',color = 'red', markersize = 2)
        plt.annotate(texte, xy = (a, float(i+1)+0.2))
    plt.axis([0, fin+2, 0,fin+2])
    plt.show()
    affichage()
```