Algorithmes d'optimisation

Version 1.3

Michaël Guedj

Table des Matières

1 Approche des problèmes d'optimisation

Nous considérons une veille radio que l'on trouve dans une brocante ; elle comporte une molette : A ; et 3 boutons : B, C et D.

- A permet de capter des fréquences (100 fréquences disponibles) ;
- B, C et D sont trois boutons qui peuvent prendre 10 valeurs chacune – mais dont on ne connait la signification.

Une solution est la donnée d'un quadruplet (a, b, c, d) où a, b, c et d indiquent des positions, de la molette et des différents boutons.

Au total : $100 \times 10 \times 10 \times 10 = 100\ 000$ solutions possibles.

Le but est de trouver une station diffusant une chanson que l'on aime bien, et avec une bonne qualité de diffusion.

Pour chaque solution, i.e. chaque possibilité de quadruplet, on donne une note.

Exemple de notation.

On entend :

- Seulement des bruits de grésillement $\rightarrow 0/20$;
- Des voix, avec beaucoup de grésillement $\rightarrow 5/20$;
- Une chanson avec grésillement $\rightarrow 12/20$;
- Des voix ; avec bonne qualité d'écoute $\rightarrow 14/20$;
- Une chanson « sympathique » ; avec bonne qualité d'écoute $\rightarrow 17/20$;
- Une chanson que l'on aime davantage ; avec bonne qualité d'écoute $\rightarrow 19/20$.

Nous nous faisons face à un problème d'**optimisation** ; il s'agit, en effet, de trouver une solution qui **maximise** la note.

Etant donné le nombre de solutions possibles ; on utilise alors une heuristique de résolution. (Tester 100 000 configurations possibles n'est pas acceptable pour un humain).

2 Retour sur trace (*Backtracking*)

Définition (problème des n reines)**.**

- Entrée.
  - Un échiquier $n \times n$.
  - n reines, issues de l'ensemble

$$Q := \{Q_i \mid i = 1, ..., n\}$$

- Problème.
  - Placer les n reines sur l'échiquier, de sorte à ce qu'aucune ne puisse en "manger" une autre.

Théorème. *La résolution du problème des n reines, par l'algorithme de retour sur trace, est en $O(n!)$.*

Preuve.

- n majore le nombre de positions de Q_1 ;
- Q_1 placée ; $n-1$ majore le nombre de positions de Q_2 ;
- Q_1 et Q_2 placées ; $n-2$ majore le nombre de positions de Q_3 ;
- ... ;
- $Q_1, ..., Q_i$ placées ; $n-i$ majore le nombre de positions de Q_{i+1} ;
- ... ;
- $Q_1, ..., Q_{n-2}$ placées ; $n-(n-2) = 2$ majore le nombre de positions de Q_{n-1} ;
- $Q_1, ..., Q_{n-1}$ placées ; $n-(n-1) = 1$ majore le nombre de positions de Q_n.

Le nombre de positions effectuées, par l'algorithme de retour sur trace, est alors majoré par $n!$. □

3 Algorithmes de type glouton

Définition (heuristique)**.** Une heuristique (du grec ancien *eurisko* « je trouve ») est une méthode de calcul qui fournit "rapidement" une solution "réalisable" (non nécessairement optimale ou exacte) pour un problème d'optimisation "difficile".

Note : Une heuristique s'impose quand les algorithmes de résolution exacte sont de complexité exponentielle, et dans beaucoup de problèmes "difficiles". L'usage d'une heuristique est également pertinent pour calculer une solution approchée d'un problème, ou encore pour "accélérer" un processus de résolution exacte. Généralement, une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre, mais peut contenir des principes plus généraux.

Définition (algorithme glouton)**.** Un algorithme glouton (*greedy algorithm* en anglais) est un algorithme qui suit le principe de faire, étape par étape, un choix optimum (local). Dans certains cas, cette approche permet d'arriver à un optimum global ; mais dans le cas général, c'est une heuristique.

4 Problème de la somme du sous-ensemble (*Subset Sum Problem*)

4.1 Définition du problème

Définition (problème de la somme du sous-ensemble)**.**

- Entrée.
  - Un tableau t, de taille n, à valeurs entières.
  - Une capacité $c \in \mathbb{N}$.
- Problème.

  Trouver k indices de t, distincts, soient $i_1, ..., i_k$, tels que la somme

  $$t[i_1] + ... + t[i_k]$$

  soit la plus grande possible, tout en assurant

  $$t[i_1] + ... + t[i_k] \leq c$$

Théorème. *Le problème de la somme du sous-ensemble est NP-hard.*

Proof. Admis. □

4.2 Heuristique gloutonne

Algorithm 1 `greedy`(t, n, c)

```
trier t selon l'ordre décroissant
res ← [ ]
val ← 0
for i ← 0, ..., n − 1 do
    if val + t[i] ≤ c then
        res ← res + [ i ]
        val ← val + t[i]
    end if
end for
return res
```

Complexité : $O(n. \log n)$.

5 Problème de la coloration de graphe (*Graph Coloring Problem*)

5.1 Définition du problème

Définition (problème de la coloration de graphe)**.**

- Entrée.

  Un graphe G, non orienté, sans boucle.

- Problème.

  Trouver le plus petit entier k, tel que G soit k-colorable.

Théorème. *Le problème de la coloration de graphe est NP-hard.*

Proof. Admis. □

5.2 Coloration gloutonne

Algorithm 2 `Greedy_Colouring`$(G = (S, A))$ tel que $S = \{s_i : i \in \{0, ..., n-1\}\}$

$\mathit{color} \leftarrow$ un dictionnaire vérifiant : $\forall s \in S, \mathit{color}[s] = -1$
for $i \leftarrow 0, .., n-1$ **do**
    $C_i \leftarrow \{\mathit{color}[s_j] : (s_i, s_j) \in A\}$
    $\mathit{color}[s_i] \leftarrow \min\{c \in \mathbb{N} : c \notin C_i\}$ ▷ Le plus petit entier positif n'appartenant pas à C_i
end for
return color

6 Problème du voyageur de commerce (*Travelling Salesman Problem*)

6.1 Définition du problème

Définition (cycle hamiltonien)**.** Un cycle hamiltonien, d'un graphe, est un cycle passant par tous les sommets du graphe, une fois, et une seule.

Définition (problème du voyageur de commerce)**.**

- Entrée.

  Un graphe $G = (S, A, cost)$, tel que :
  - G est non orienté ;
  - G est complet ;
  - Les arcs de G sont valués par la fonction $cost : A \to \mathbb{R}$.

- Problème.

  Trouver le cycle hamiltonien ayant le coût le plus faible.

Théorème. *Le problème du voyageur de commerce est NP-hard.*

Proof. Admis. □

6.2 Heuristique : algorithme du plus proche voisin

Algorithm 3 `nearest_neighbour`$(G = (S, A, cost))$

$todo \leftarrow \{\}$
$res \leftarrow$ `[ ]`
$s_0 \leftarrow$ choisir un sommet arbitraire de G
$todo \leftarrow G - \{s_0\}$
$res \leftarrow res +$ `[` s_0 `]`
$s \leftarrow s_0$
while $todo \neq \{\}$ **do**
    $s' \leftarrow$ le sommet le plus proche de s
    $todo \leftarrow todo - \{s'\}$
    $res \leftarrow res +$ `[` s' `]`
    $s \leftarrow s'$
end while
return $res \leftarrow res +$ `[` s_0 `]`

7 Algorithmes génétiques

On cherche à maximiser $f : E \to \mathbb{R}$; où E est fini.

Exemples d'ensembles E (où n est un entier) :

- $E = \texttt{float}^n$;
- $E \subset \texttt{float}^n$;
- $E = \mathbb{B}^n$;
- $E = S_n$ (l'ensemble des permutations de $\{1, ..., n\}$).

Algorithm 4 `algorithme_genetique`(n : taille d'un individu, p : nombre d'individus, ...)

$P \leftarrow$ Population initiale
$res \leftarrow$ `None` ▷ en considérant $f(\texttt{None}) = -\infty$
while **not** (Critère Fin) **do**
  Calculer la valeur de fitness de chaque individu ▷ i.e., $f(p), \forall p \in P$
  $best \leftarrow$ le meilleur individu de P ▷ au sens de la maximisation de f
  if $f(best) > f(res)$ **then**
    $res \leftarrow best$
  end if
  $P \leftarrow \texttt{selection}(P)$
  $P_{new} \leftarrow \texttt{reproduction}(P)$
  $P_{new} \leftarrow \texttt{mutation}(P_{new})$
  $P \leftarrow P_{new}$
end while
return res

(Les phases de *selection* et de *reproduction* peuvent être regroupées en une routine).