IFT2125 - Introduction à l'algorithmique

Algorithmes voraces (gloutons) (B&B chapitres 5-6)

Greedy algorithms

Pierre McKenzie

DIRO, Université de Montréal

Automne 2017

Structures de données

Data Structures

À parcourir par vous-mêmes si cette matière est nouvelle pour vous :

To go through yourselves if this material is new to you:

Chapitre 5, Data structures

- Section 5.1 : Arrays, stacks and queues
- Section 5.2 : Records and pointers
- Section 5.3 : Lists
- Section 5.4 : Graphs
- Section 5.5 : Trees
- Section 5.7 : Heaps

are to be read on your own if you have not come across that material before.

Les algorithmes voraces (greedy algorithms)

Our first large class of algorithms

- Notre première grande classe d'algorithmes.
- S'applique à une forme de problèmes d'optimisation.

Applies to some form of optimization problems.

Le type de problème à résoudre

The type of problem to be solved

Identifier, parmi un ensemble de candidats disponibles, un sous-ensemble formant une solution, optimale parmi les solutions possibles.

Identify, among a set of available candidates, a subset forming an optimal solution among the possible solutions.

Exemples :

Set of maximum weight of linearly independent vectors

- Ensemble de poids maximum de vecteurs linéairement indépendants

Set of edges forming a minimal spanning tree

- Ensemble d'arêtes formant un arbre sous-tendant minimal

The fewest possible coins totaling a given amount

- Le moins de pièces de monnaie possible totalisant un montant donné

Maximum value of price to be carried when a jewelry is stolen

- Valeur maximale du butin à emporter lors du vol d'une bijouterie

Path of minimum length between 2 vertices of a graph

- Chemin de longueur minimale entre 2 sommets d'un graphe

Minimum average time for n tasks on a processor

- Temps moyen minimum pour n tâches sur un processeur

Maximum 3-colorable subgraph of a graph

- Sous-graphe 3-coloriable maximal d'un graphe

The idea is simple: we are hungry or eaten

L'idée est simple : on a faim on bouffe !

Un candidat ajouté n'est jamais écarté ensuite

An added candidate is never dismissed

function *greedy*(C: *set*): *set*
  {C is the set of candidates}
  $S \leftarrow \varnothing$ {We construct the solution in the set S}
  while $C \neq \varnothing$ **and not** *solution*(S) **do**
    $x \leftarrow$ *select*(C)
    $C \leftarrow C \setminus \{x\}$
    if *feasible*($S \cup \{x\}$) **then** $S \leftarrow S \cup \{x\}$
  if *solution*(S) **then return** S
    else return "*there are no solutions*"

Composantes fréquentes d'un algorithme vorace

Fonction de sélection

Fonction objective : est-ce que S forme déjà une solution?

function *greedy*(C: *set*): *set*
    {C is the set of candidates}
    $S \leftarrow \emptyset$ {We construct the solution in the set S}
    while $C \neq \emptyset$ **and not** *solution*(S) **do**
        $x \leftarrow$ *select*(C)
        $C \leftarrow C \setminus \{x\}$
        if *feasible*($S \cup \{x\}$) **then** $S \leftarrow S \cup \{x\}$
    if *solution*(S) **then return** S
                    else return "*there are no solutions*"

Fonction de réalisabilité

Ensemble de Candidats choisis

Composantes

The selection function:
La fonction de sélection :

- Souvent le prochain candidat de la liste triée des candidats non encore considérés
  Often the next candidate from the sorted list of candidates not yet considered

The feasibility function:
La fonction de réalisabilité :

- Centrale à l'algorithme Central to the algorithm
- Son implantation détermine souvent la performance
  Its implementation often determines performance

The objective function:
La fonction objective :

- Souvent naturelle et facile à déduire du problème
  Often natural and easy to deduce from the problem

Exemple

data vectors with weight

Donnée : vecteurs $v_1, \ldots, v_m \in \mathbb{R}^d$, avec poids $P[1], \ldots, P[m] \in \mathbb{R}^+$

Déterminer : ensemble de vecteurs lin. indép. de poids total maximum

Determine : set of lin. indep. vectors of maximum total weight

candidates: vectors

- candidats : vecteurs

feasibility function linear independence test

- fonction de réalisabilité : test d'indépendance linéaire
- fonction de sélection $=$?

selection function

Exemple (suite)

L'algorithme

set of vectors

fonction *Vecteurs*(V, $P[1..|V|]$) : ensemble de vecteurs

  { $P[a]$ est le poids du vecteur a } P[a] is the weight of the vector a

  L[1..|V|] $\leftarrow$ liste des vecteurs triée en ordre de poids décroissants

list of vectors sorted in descending order of weight

  $S \leftarrow \emptyset$

  tantque L non vide **faire** while L is not empty

    $a \leftarrow$ retirer prochain vecteur de L remove the next vector of L

    si $S \cup \{a\}$ lin. indép. **alors** $S \leftarrow S \cup \{a\}$

if S ∪ {a} lin. indep. then

  retourner S

return

Returns a set of linearly independent vectors of maximum weight

- Retourne ensemble de vecteurs linéairement indépendants de poids maximum ?
- Aucune fonction objective requise, ok ?

  No objective function required, ok?

Let C be a finite set of candidates and I ⊆ {S : S ⊆ C}

Soit C un ensemble fini de candidats et $I \subseteq \{S : S \subseteq C\}$

- i.e., $I = \{S_1, \ldots, S_k\}$ where the où les $S_i \subseteq C$ ~~we call "Independent" the elements of I~~ we call the elements of I is "Independent"
- on appelle "indépendants" les éléments de I
  - penser "indépendant" = ensemble de vecteurs lin. indép. think of the linearly independent columns of a matrix

(C, I) is a matroid iff for every

(C, I) est un matroïde ssi pour tout $X, Y \subseteq C$:

- la trivialité : $\emptyset \in I$ triviality : ∅ ∈ I
  - l'ensemble vide est indépendant
- l'hérédité : $[X \in I \text{ et } Y \subseteq X] \Rightarrow Y \in I$ heredity : [X ∈ I and Y ⊆ X] ⇒ Y ∈ I
  - un sous-ensemble d'un indépendant est indépendant
    a subset of an independent set is independent
- l'échange : $[X \in I \text{ et } Y \in I \text{ et } |X| < |Y|] \Rightarrow (\exists a \in Y \setminus X)[X \cup \{a\} \in I]$.
  the exchange:
  [X ∈ I and Y ∈ I and |X| < |Y|] ⇒ (∃a ∈ Y \ X)[X ∪ {a} ∈ I]
  - on peut toujours ajouter un candidat à un ensemble indépendant X si la size taille d'un indépendant Y dépasse celle de X
    we can always add a candidate to an independent set X if the size of an independent Y exceeds that of X

Matroïde

Exemples

boat, apple, avocado, bicycle, cellular, screwdriver

1. $C = \{$ bateau, pomme, avocat, vélo, cellulaire, tournevis $\}$
   I l'ensemble des sous-ensembles de C possédant 3 éléments ou moins
   the set of subsets of C having 3 elements or less

   set of vectors

2. C ensemble de vecteurs
   I l'ensemble des sous-ensembles de C linéairement indépendants
   the set of linearly independent subsets of C

   the set of edges of a given graph

3. C l'ensemble des arêtes d'un graphe donné
   I l'ensemble des sous-ensembles d'arêtes ne créant pas de cycle
   the set of sub-sets of edges, not creating a cycle

Let (C, I) be a matroid

Soit (C, I) un matroïde.

fact

Fait

All the S ∈ I of maximal cardinality have the same number of elements.

1. *Tous les $S \in I$ de cardinalité maximale ont même nombre d'éléments.*
2. *Pour tout $B \subseteq C$, $(B, \{B \cap S : S \in I\})$ est un matroïde.*

For all B ⊆ C, (B, {B ∩ S: S ∈ I}) is a matroid

Ensemble indépendant de poids maximum d'un matroïde

Maximum weight independent set of a matroid

data · is matroid · each candidate of C having a weight

Donnée : (C, I) un matroïde, chaque candidat de C ayant un poids $\in \mathbb{R}^+$

Déterminer : $S \in I$ de poids maximum

Determine : S ∈ I of maximum weight

candidates : elements of C

- candidats : éléments de C

feasibility function : current set belongs to I?

- fonction de réalisabilité : ensemble courant appartient à I ?
- fonction de sélection $=$ plus lourd candidat non encore considéré

selection function = heavier candidate not yet considered set of cadidates

fonction *MaxIndép*$(C, I, P[1..|C|])$: ensemble de candidats

 $\{$ $P[a]$ est le poids du candidat a $\}$ P[a] is the weight of candidate a

 L[1..|C|] $\leftarrow$ liste des candidats triée en ordre de poids décroissants

 $S \leftarrow \emptyset$ list of candidates sorted in descending order of weight

 while L is not empty

 tantque L non vide **faire**

  remove next candidate of L

  $a \leftarrow$ retirer prochain candidat de L

  if then

  si $S \cup \{a\} \in I$ **alors** $S \leftarrow S \cup \{a\}$

 retourner S

 return

Returns an element of I of maximum weight? Proof to be made

- Retourne un élément de I de poids maximum ? Preuve à faire.
- Intérêt ? Interest?
  applies to more than one situation
  - s'applique à plus d'une situation
  - capture bien la "voracité"
    captures the "greedy"

data / vectors / with weight

Donnée : vecteurs $v_1, \ldots, v_m \in \mathbb{R}^d$, avec poids $P[1], \ldots, P[m] \in \mathbb{R}^+$

Déterminer : ensemble de vecteurs lin. indép. de poids total maximum

Determine : set of lin. indep. vectors of maximum total weight

consider

- considérer
  $C = \{v_1, \ldots, v_m\}$ the vectors of S are linearly independent
  $I = \{S \subseteq C :$ les vecteurs de S sont linéairement indépendants $\}$
- *MaxIndép*$(C, P[1..m])$ est précisément notre algorithme *Vecteurs*$(V, P[1..|V|])$ is precisely our algorithm
- suffit donc de vérifier que (C, I) est un matroïde
  is sufficient to verify that (C, I) is a matroid

Minimum spanning tree

Arbres sous-tendants (couvrants) minimaux

Une autre application des matroïdes

Another application of matroids

Let (N, A) be a graph and I = {S ⊆ A | (N, S) is cycleless (acyclic).

Soit (N, A) un graphe et $I = \{S \subseteq A \mid (N, S) \text{ est sans cycle}\}$.

Proposition

(A, I) est un matroïde.

(A, I) is a matroid.

Arbres sous-tendants minimaux (suite)

Corollaire

On peut utiliser MaxIndép pour calculer un arbre sous-tendant minimal d'un graphe connexe avec poids $\in \mathbb{R}^+$ *aux arêtes.*

MaxIndep can be used to compute a minimum spanning tree of a connected graph with weight ∊ R+ at the edges.

Proof :

Preuve :

Minimum spanning tree (cont.)

Arbres sous-tendants minimaux (suite)

Corollaire

On peut utiliser MaxIndép *pour calculer un arbre sous-tendant minimal d'un graphe connexe avec poids* $\in \mathbb{R}^+$ *aux arêtes.*

MaxIndep can be used to compute a minimum spanning tree of a connected graph with weight ∊ R+ at the edges.

Proof :

Preuve :

let m = max {weight (a) I a ∊ A}

- soit $m = \max\{\text{poids}(a) \mid a \in A\}$

associating to each a ∊ A the new weight 'm - weight(a)'

- associer à chaque $a \in A$ le nouveau poids $m - \text{poids}(a)$

an independent S of maximal cardinality will necessarily be a tree underlying and will include I A I - 1 edge

- un indépendant S de cardinalité maximale sera forcément un arbre sous-tendant et comprendra $|A| - 1$ arêtes
- S aura maximisé $(m \times |S|) -$ "poids total de S

S will maximize : (m × ISI) - (total weight of S)

- S aura donc minimisé poids total de S

S will thus have minimized total weight of S

Maxindép devient alors l'algorithme de Kruskal !

Maxindep then becomes the Kruskal algorithm!

function *Kruskal*($G = \langle N, A \rangle$: *graph*; *length*: $A \to \mathbb{R}^+$): *set of edges*
    {initialization}
    Sort A by increasing length
    $n \leftarrow$ the number of nodes in N
    $T \leftarrow \varnothing$ {will contain the edges of the minimum spanning tree}
    Initialize n sets, each containing a different element of N
    {greedy loop}
    repeat
        $e \leftarrow \{u, v\} \leftarrow$ shortest edge not yet considered
        ucomp $\leftarrow$ *find*(u)
        vcomp $\leftarrow$ *find*(v)
        if *ucomp* $\neq$ *vcomp* **then**
            merge(*ucomp*, *vcomp*)
            $T \leftarrow T \cup \{e\}$
    until T contains $n - 1$ edges
    return T

- Candidats = arêtes   Candidates = edges
- Réalisabilité : arête relie des composantes connexes distinctes ?
  Feasibility : Edge connects disjoint connected components?
- Fonction objective : $n - 1$ arêtes choisies ?
  Objective function : n - 1 selected edges?

Autre exemple vorace : remise de la monnaie

Another greedy example: delivery of the currency

Autre exemple vorace : remise de la monnaie

Another greedy example: delivery of the currency

```
function {make-change}(n): set of coins
    {Makes change for n units using the least possible
     number of coins. The constant C specifies the coinage}
    const C = {100, 25, 10, 5, 1}
    S ← ∅ {S is a set that will hold the solution}
    s ← 0 {s is the sum of the items in S}
    while s ≠ n do
        x ← the largest item in C such that s + x ≤ n
        if there is no such item then
            return "no solution found"
        S ← S ∪ {a coin of value x}
        s ← s + x
    return S
```

Mais attention !

But beware !

A greedy heuristic can be applied to a problem without constituting a correct voracious algorithm

- Une heuristique vorace peut s'appliquer à un problème sans constituer un algorithme vorace correct
  - par exemple, si l'heuristique ne garantit pas que la solution trouvée vérifie le critère d'optimalité

    for example, if the heuristic does not guarantee that the resulting solution satisfies the criterion of optimality

Au fait, l'approche vorace fonctionne-t-elle au Canada (avec ou sans la "penny" d'avant 2013) pour la remise de la monnaie ?

By the way, does the greedy approach work in Canada (with or without the pre-2013 "penny") for the sum of the money?

© J.J.'s Complete Guide to Canada.

Au fait, l'approche vorace fonctionne-t-elle au Canada (avec ou sans la "penny" d'avant 2013) pour la remise de la monnaie ?

By the way, does the greedy approach work in Canada (with or without the pre-2013 "penny") for the sum of the money?

© J.J.'s Complete Guide to Canada.

Yes ! But provided that an unlimited number of pieces of each denomination are available!

Oui ! Mais pourvu qu'un nombre illimité de pièces de chaque dénomination soient disponibles !

Remise de la monnaie

Remittance of the currency (sum of the money)

Does the voracious approach work in all countries when an unlimited number of coins are available?

L'approche vorace fonctionne-t-elle dans tous les pays lorsqu'un nombre illimité de pièces est disponible ?

Probably, but not in general!

Probablement, mais pas en général ! (fails if the coin designs don't have a coin for 1.)

Fails for some coin designs!

- Échoue pour certaines dénominations de pièces !
- Étonnamment difficile de démontrer le bon fonctionnement, même pour les pièces canadiennes

  Surprisingly difficult to demonstrate proper operation, even for Canadian parts

Retour à l'exemple

function {*make-change*}(n): *set of coins*
    {Makes change for n units using the least possible
    number of coins. The constant C specifies the coinage}
    const $C = \{100, 25, 10, 5, 1\}$
    $S \leftarrow \emptyset$ {S is a set that will hold the solution}
    $s \leftarrow 0$ {s is the sum of the items in S}
    while $s \neq n$ **do**
        $x \leftarrow$ the largest item in C such that $s + x \leq n$
        if there is no such item **then**
            return "*no solution found*"
        $S \leftarrow S \cup$ {a coin of value x}
        $s \leftarrow s + x$
    return S

Here the candidates are implicit

- Ici les candidats sont implicites

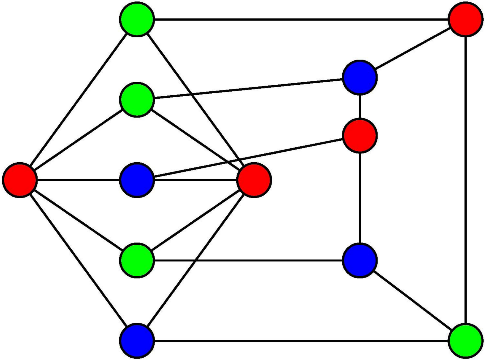
feasibility function: s + x ≤ n, i.e., "does the addition of x to the selected candidates (total) add up at most n?"

- Fonction de réalisabilité : $s + x \leq n$, i.e., "est-ce que l'ajout de x aux candidats choisis totalise au plus le montant n à remettre ?"
- Fonction objective : $s = n$, i.e., "est-ce que les pièces totalisent exactement n ?"

Objective function: s = n, i.e., "do the coins add up exactly n?

greedy approach does not work to solve 3-colorability

L'approche vorace ne fonctionne pas pour résoudre la 3-colorabilité



Will solve some but not all copies

- Résoudra certains exemplaires mais pas tous
- Peut servir d'heuristique en vue d'une solution approchée

  Can be used as a heuristic for an approximate solution

This notion (greedoid) exists

Cette notion (greedoid) existe !

Mais nous ne l'étudierons pas (au-delà de sa définition).

But we will not study it (beyond its definition).

(C, I) is a gluttonoid iff for all X, Y ⊆ C :

(C, I) est un gloutonnoïde ssi pour tout $X, Y \subseteq C$:

- la trivialité : comme avant. the triviality: as before.
- accessibility
  l'accessibilité : $\emptyset \neq X \in I \Rightarrow \exists a \in X, X \setminus \{a\} \in I$
- l'échange : comme avant.
  the exchange: as before.

On peut démontrer... We can demonstrate ...

Voir par exemple dans Handbook of Combinatorics, vol. 1
See, for example, Handbook of Combinatorics, Vol. 1

C ensemble fini de candidats et $I \subseteq 2^C$
C finite set of candidates and I ⊆ 2C

Théorème (du matroïde) Theorem (of matroid)

Let (C,I) follow the triviality and the heredity.
Soit (C, I) muni de la trivialité et de l'hérédité.

MaxIndép obtient $S \in I$ de poids maximal ssi (C, I) possède l'échange.

MaxIndep obtains S ∈ I of maximum weight iff (C, I) possesses the exchange.

Théorème (du gloutonnoïde) Theorem (of gluttonoid)

Let (C,I) follow the triviality and the accessibility.
Soit (C, I) muni de la trivialité et de l'accessibilité.

MaxIndép obtient $S \in I$ de poids maximal ssi (C, I) possède l'échange.
MaxIndep obtains S ∈ I of maximum weight iff (C, I) possesses the exchange.

Corollaire

L'algorithme de Prim qui suit calcule un arbre sous-tendant minimal.

The following Prim algorithm computes a minimal spanning tree.