RÉVISIONS INFORMATIQUE

Algorithmes au programme

Hugo Salou MPI*

Table des matières

Algorithmes manquants

Apprentissage Arbres k-dimensionnels.

Recherche textuelle. Codage HUFFMAN, LZW, BOYER-MOORE, RABIN-KARP.

Graphes. DIJKSTRA, $A^\star$ et conversion en arbre binaire.

Jeux. MINMAX et calcul d'attracteurs.

Autres. Tri par tas, tri fusion et dichotomie.

Non vus. PETERSON et LAMPORT.

1 Logique

Idée de l'algorithme de QUINE

Pour trouver un modèle de G (sous CNF), on choisit une variable propositionnelle p (au hasard ou par contrainte) et, d'une part, on tente de résoudre avec p vraie, sinon, on résout avec p faux.

Algorithme 1 Algorithme de QUINE

Procédure *Assume*(H, p, b)
  si $b = \boldsymbol{F}$ **alors**
    On pose $\ell_{\boldsymbol{V}} = p$ et $\ell_{\boldsymbol{F}} = \neg p$.
  sinon
    On pose $\ell_{\boldsymbol{V}} = \neg p$ et $\ell_{\boldsymbol{F}} = p$.
  pour C une clause de H **faire**
    si $\ell_{\boldsymbol{V}} \in C$ **alors** On retire C de H.
    sinon si $\ell_{\boldsymbol{F}} \in C$ **alors** On retire $\ell_{\boldsymbol{F}}$ de C.

si G est vide **alors retourner** OUI
si G contient la clause vide **alors retourner** NON
si il existe une clause contenant un seul littéral ℓ **alors**
  si $\ell = p$ **alors retourner** QUINE(ASSUME$(G, p, \boldsymbol{V})$)
  sinon retourner QUINE(ASSUME$(G, p, \boldsymbol{F})$)
sinon
  On choisit une variable p de G.
  retourner QUINE(ASSUME$(G, p, \boldsymbol{V})$) OU QUINE(ASSUME$(G, p, \boldsymbol{F})$)

2 Apprentissage – *Machine learning*

Idée de l'algorithme des k plus proches voisins

Les k voisins proches d'un point $\bar{v} \in \mathbb{R}^n$ permettent de déterminer sa classification. On note la classe des voisins proches, et on choisit la plus présente.

Algorithme 2 k plus proches voisins

Entrée (S, c) un jeu de données classifié

1: On trie, en $(p_i)_{i \leqslant N}$ les données par distance croissante à $\bar{v}$.
2: Soit D un dictionnaire où, $\forall x \in \mathscr{C}$, $D[x] = 0$.
3: **pour** $j \in [\![1, k]\!]$ **faire**
4: $\quad D[c(p_j)] \leftarrow D[c(p_j)] + 1$
5: **retourner** $\arg\max_{d \in \mathscr{C}} D[d]$

Idée des arbres k dimensionnels

À faire…

Idée de l'algorithme ID3

Pour chaque critère, on calcule l'entropie H, et on choisit celui avec la plus basse. Ceci génère deux branches sur lesquelles on peut itérer l'algorithme. L'entropie d'une partition $(S_i)_{i \in [\![1,n]\!]}$ d'un jeu de données classifié (S, c) est

$$\mathrm{H}\big((S_i)_{i \in [\![1,n]\!]}, c\big) = \sum_{i=1}^{n} \frac{\#S_i}{\#S} \cdot \mathrm{H}(S_i, c),$$

où $\mathrm{H}(S_i, c)$ est l'entropie d'un ensemble est

$$\mathrm{H}(S, c) = -\sum_{\text{classe } e} \frac{\#\text{classés } e}{\#S} \ln\left(\frac{\#\text{classés } e}{\#S}\right).$$

Idée de l'algorithme HAC

On commence avec un élément par classe. Les classes minimisant la mesure de dis-similarité choisie sont fusionnées.

Algorithme 3 Algorithme HAC – classification hiérarchique ascendante

$P \leftarrow \{\{x\} \mid x \in \mathscr{D}\}$
tant que $\#P \geqslant 2$ et ⟨critère⟩ **faire**
    Soient (A, B) minimisant $D(A, B)$
    $P \leftarrow (P \setminus \{A, B\}) \cup \{A \cup B\}$
retourner P

Idée de l'algorithme k-moyenne

On fixe k, le nombre de classes. On cherche à fixer des « représentants » (les vecteurs m_i) : pour chaque vecteur de $\mathscr{D}$, sa classe correspond à celle de son représentant le plus proche. À chaque itération, on calcule le barycentre des éléments de même classe, et on change le représentant associé.

Algorithme 4 k-moyenne

$(m_i)_{i\in[\![1,k]\!]} \leftarrow k$ vecteurs de $\mathscr{D}$
stable $\leftarrow \boldsymbol{F}$
tant que ¬stable **faire**
    $C \leftarrow$ le partitionnement des représentants les plus proches
    $m' \leftarrow (\text{barycentre}(C_i))_{i\in[\![1,k]\!]}$ ▷ *Re-calcul des barycentres*
    si $m' = m$ **alors** stable $\leftarrow \boldsymbol{V}$
    sinon $m \leftarrow m'$
retourner C.

3 Graphes et arbres

3.1 Parcours

Parcours d'arbres binaires

Les parcours sont réalisés dans les ordres suivants :

— *préfixe* : racine, fils gauche, fils droit ;
— *infixe* : fils gauche, racine, fils droit ;
— *suffixe* : fils gauche, fils droit, racine.

Parcours de graphes

Parcours en largeur. On parcourt le graphe par "couches" : à chaque étape, on "note" les successeurs non visités des sommets visités, et on les parcourt un à un ; puis, on répète.

Parcours en profondeur. On parcourt le graphe "la tête la première" : à chaque étape, on visite le premier successeur non visité du dernier sommet visité ; quand il n'en a plus, on remonte au visité précédent.

3.2 Composantes (fortement) connexes

Composantes connexes dans un graphe non orienté

On choisit un sommet v du graphe **non orienté**. On parcourt, en largeur ou en profondeur, les sommets accessibles depuis v. À la fin du parcours, on trouve la composante connexe dans laquelle est v. On recommence avec les sommets non parcourus.

Idée de l'algorithme de Kosaraju – recherche de CFC

Pour trouver les CFC, on réalise un parcours en profondeur (à l'aide d'un tri préfixe), on inverse les arêtes, et on réalise un nouveau parcours en profondeur (avec ce tri préfixe).

Algorithme 5 Algorithme de Kosaraju – recherche de CFC dans un graphe orienté

```
1:  Procédure TriPréfixe
2:      P ← (  )        ▷ Pile vide
3:      tant que S \ P ≠ ∅ faire
4:          Soit v ∈ S \ P.
5:          On réalise un parcours en profondeur depuis v.
6:          On empile dans P les sommets visités.
7:      retourner P

8:  P ← TriPréfixe( )
9:  On "transpose" le graphe en G^⊤.
10: tant que P ≠ ∅ faire
11:     On dépile v de P.
12:     On réalise un parcours en profondeur depuis v dans G^⊤.
13:     pour tout sommet s visité faire
14:         On supprime s de P.
15:     Les sommets visités forment la CFC de v.
```

3.3 Arbres couvrants de poids minimal

Idée de l'algorithme de KRUSKAL

On ajoute à l'arbre les arêtes de poids minimal sans créer de cycles.

Algorithme 6 Algorithme de KRUSKAL – recherche d'arbre couvrant de poids minimal

1: $B \leftarrow \varnothing$. ▷ *Arêtes de l'arbre couvrant*
2: **tant que** $\exists (u, v)$, $u \nsim_B v$ **faire**
3: Soit $\{x, y\}$ un arête de poids minimal avec $x \nsim_B y$.
4: $B \leftarrow B \cup \{\{x, y\}\}$.
5: **retourner** (S, B), un arbre couvrant de poids minimal.

Amélioration de KRUSKAL avec une structure UnionFind

Comme l'objectif est de trouver un arbre, on peut initialiser une partition d'une structure UnionFind où chaque sommet est seul, et on réalise $n - 1$ `unions` en suivant les arêtes de poids minimal.

3.4 Couplages

Idée du calcul de chaîne augmentante

On rappelle qu'une chaîne augmentante est une chaîne alternée dont les deux extrémités sont libres. On part d'un sommet initial. On commence avec une chaîne contenant uniquement ce sommet initial. Tant que ce sommet a des successeurs qui ne sont pas dans la chaîne, on l'ajoute s'il est libre ; sinon, on augmente la chaîne.

Algorithme 7 Calcul d'une chaîne augmentante à partir d'un sommet $s \in S$.

1: **Procédure** AUGMENTE(x, chaîne)
2: **pour** $y \in \mathrm{Succ}(x) \setminus$ chaîne **faire**
3: **si** y est libre dans C **alors**
4: **retourner** $\mathrm{Some}(\text{chaîne} \uplus (y))$
5: **sinon**
6: Soit z tel que $\{y, z\} \in C$.
7: **retourner** AUGMENTE(z, chaîne $\uplus (y, z)$)

8: **si** s est libre dans C **alors**
9: **retourner** AUGMENTE($s, (s)$)
10: **sinon**
11: **retourner** None

Autre idée du calcul d'une chaîne augmentante

Soit $G = (U \uplus V, A)$ un graphe biparti. On oriente les arêtes non libres de V vers U, et les arêtes libres de U vers V. Une chaîne augmentante est un chemin d'un sommet libre de U vers un sommet libre de Y. On peut en calculer une à l'aide d'un parcours de ce graphe en largeur ou en profondeur.

Idée du calcul d'un couplage maximum

On commence avec un couplage vide. Tant que l'on peut y trouver une chaîne augmentante, il n'est pas maximum, on l'inverse et on recommence.

Algorithme 8 Calcul d'un couplage maximum

Entrée $G = (S, A)$ un graphe biparti, avec $S = S_1 \uplus S_2$

$C \leftarrow \varnothing$
Done $\leftarrow \varnothing$
tant que $\exists x \in S_1 \setminus \text{Done}$ **faire**
    $r \leftarrow$ CHAÎNE-AUGMENTANTE(G, C, x)
    si $r \neq \text{None}$ **alors**
        $\text{Some}(a) \leftarrow r$
        On inverse la chaîne a dans C.
    Done $\leftarrow \{x\} \cup$ Done
retourner C

3.5 Plus court chemin

Idée de l'algorithme de FLOYD-WARSHALL

Si aller d'un sommet i à j est plus court en passant par k, alors passons par k.

Algorithme 9 Algorithme de FLOYD-WARSHALL

Soit M une matrice initialisée à $+\infty$, de taille $n \times n$, où $n = |V|$. ▷ *Longueur des chemins : à la i-ème ligne, et à la j-ème colonne, on a la distance du chemin de i à j.*
pour toute arête $\{i, j\}$ **faire**
    $M[i, j] = p_{i,j}$, le poids de l'arête.
pour tout sommet i **faire**
    $M[i, i] = 0$.
pour tout sommet de départ i **faire**
    pour tout sommet d'arrivée j **faire**
        pour tout sommet intermédiaire k **faire**
            si $M[i, j] \geqslant M[i, k] + M[k, j]$ **alors**
                $M[i, j] \leftarrow M[i, k] + M[k, j]$.
retourner M.

À faire : DIJKSTRA et $A^\star$.

4 Automates

Déterminisation

L'automate déterminisé est construit de telle sorte que ses états sont des ensembles d'états de l'automate non-déterministe initial. Les transitions de l'automate déterministe sont déduites de la table de transitions de l'automate initial.

Suppression des ε-transitions [1]

Lors de la lecture d'une lettre, on « transitionne » vers les états pouvant être accedés à partir d'un état accessible via ε-transition. On a

$$\delta' = \{(p, a, p') \mid a \in \Sigma^\star, p \xrightarrow{\varepsilon} q \xrightarrow{a} p'\}.$$

⚠ ATTENTION. L'automate obtenu n'est pas forcément déterministe.

expression régulière $\underset{\text{BERRY-SETHI}}{\overset{\text{élimination d'états}}{\longleftrightarrow}}$ automate reconnaissant

1. cet algorithme est différent de celui vu en cours

Algorithme de Berry-Sethi

Trouvons un automate reconnaîssant $e \in \mathrm{Reg}(\Sigma)$. On « numérote » les lettres de e à l'aide d'une fonction φ ; on crée ainsi $f \in \mathrm{Reg}(\Sigma)$. On en déduit inductivement les valeurs de Λ, S, P et F du langage local de f. On fabrique un automate local reconnaîssant le langage de f, et on « dénumérote » cet automate.
Pour mieux comprendre : voir l'exemple de la sous-section 6.4 du chapitre 1 (pages 67–68 du cours complet).

Algorithme d'élimination d'états

À partir d'un automate $\mathcal{A}$, trouvons une expression régulière équivalente. On « détoure » l'automate : on définit les états q_i et q_f, seuls états initiaux et finaux de l'automate (on utilise des ε-transitions). Pour chaque état $q \in Q$, on supprime les transitions associées à q puis on supprime q.

Supprimer les transitions associées à un état q. Pour tout état $p \neq q$, s'il y a deux transitions $p \xrightarrow{r} q$ et $p \xrightarrow{r'} q$ distinctes, alors on la remplace par $p \xrightarrow{r|r'} q$.

Supprimer un état q. Si la suite de transitions $p \xrightarrow{r_1} q \xrightarrow{r_2} p'$ est valide, alors

- si la transition $q \xrightarrow{r} q$ est valide, on remplace les trois transitions par
$$p \xrightarrow{r_1 \cdot r^\star \cdot r_2} p' ;$$
- sinon, on remplace les deux transitions par $p \xrightarrow{r_1 \cdot r_2} p'$.