DS4-Durée 4h

17 janvier 2024

Ce sujet est composé de deux parties indépendantes. Le premier problème porte sur les automates et la NP-complétude alors que le second porte sur la NP-complétude et les algorithmes d'approximation.

1 MOTS SYNCHRONISANTS

1.1 MOTS SYNCHRONISANTS (EN C)

On appelle **machine** un triplet (Q, Σ, δ) où Q est un ensemble fini non vide d'**états**, Σ un alphabet et δ une application de $Q \times \Sigma$ dans Q appelée **fonction de transition**. Une machine peut donc être vue comme un automate fini déterministe complet sans notion d'état initial ou final. Comme pour les automates finis, on utilisera la notion de fonction de transition étendue définie par :

— pour tout $q \in Q$, $\delta^*(q, \varepsilon) = q$;

— pour tous $q \in Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$.

Un mot $u \in \Sigma^*$ est dit **synchronisant** pour une machine (Q, σ, δ) s'il existe $q_0 \in Q$ tel que $\forall q \in Q$, $\delta^*(q, u) = q_0$. L'existence de tels mots permet de ramener une machine dans un état particulier connu en lisant un mot donné, donc en pratique de réinitialiser une machine réelle. La figure 1 représente une machine M_0. On pourra remarquer que ba et bb sont des mots synchronisants pour M_0.

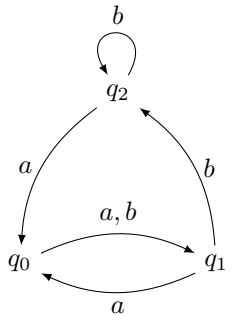

FIGURE 1 – La machine M_0.

On représente une lettre de Σ en C par un objet de type `char`. On supposera que Σ est constitué de lettres minuscules consécutives de l'alphabet courant commençant par la lettre `'a'`. Ainsi, si `a` est un objet de type `char`, alors `a - 97` est un entier compris entre 0 et 25. Un mot de Σ^* sera alors représenté par une chaîne de caractères de type `char*`. On rappelle que le caractère de fin de chaîne est `'\0'`.

Une machine sera représentée par un objet de type `machine` défini par :

```
struct Machine{
    int tQ;
    int tSigma;
    int** delta;
};

typedef struct Machine machine;
```

tel que si $M = (Q, \Sigma, \delta)$ est une machine représenté par un objet `M` de type `machine*`, alors :

— `M->tQ` représente $|Q|$, on suppose $Q = [\![0, |Q| - 1]\!]$;

— `M->tSigma` représente $|\Sigma|$, on suppose $\Sigma = [\![0, |\Sigma| - 1]\!]$;

— `M->delta` correspond à un tableau des transitions, c'est-à-dire tel que si $q \in Q$ et $a \in \Sigma$, alors `M->delta[q][a]` vaut $\delta(q, a)$.

1. Écrire une fonction `machine* init_machine(int tQ, int tSigma)` qui crée une machine telle que $|Q|$ et $|\Sigma|$ sont donnés en arguments et, pour tout $q \in Q$ et $a \in \Sigma$, $\delta(q, a) = q$.
2. Écrire une fonction `void liberer_machine(machine* M)` qui libère l'espace mémoire occupé par une machine.
3. Que dire de l'ensemble des mots synchronisants pour une machine ayant un seul état ?

   Dans toute la suite du problème, on supposera que les machines ont au moins deux états.

4. On considère la machine M_1 représentée figure 2, sur l'alphabet $\Sigma = \{a\}$. Donner un mot synchronisant pour M_1 s'il en existe un. Justifier la réponse.

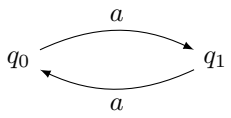

FIGURE 2 – La machine M_1.

5. Donner un mot synchronisant de trois lettres pour la machine M_2 représentée figure 3. On ne demande pas de justifier la réponse.

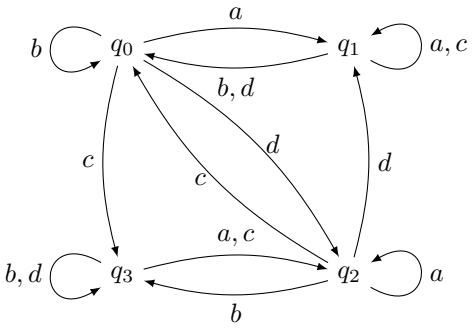

FIGURE 3 – La machine M_2.

6. Écrire une fonction `int delta_etoile(machine* M, int q, char* u)` qui prend en arguments une machine $M = (Q, \Sigma, \delta)$, un état q et un mot u et renvoie $\delta^*(q, u)$. On prendra garde que `u` est une chaîne de caractères, et qu'un caractère a une valeur comprise entre 97 et 122 et non entre 0 et 25.
7. Écrire une fonction `bool synchronisant(machine* M, char* u)` qui prend en argument une machine M et un mot u et renvoie le booléen `true` si u est synchronisant pour M et `false` sinon.
8. Montrer que si une machine admet un mot synchronisant alors il existe $a \in \Sigma$ et $q \neq q' \in Q$ tels que $\delta(q, a) = \delta(q', a)$.

   Soit $LS(M)$ le langage des mots synchronisants d'une machine $M = (Q, \Sigma, \delta)$. On introduit la machine des parties $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta})$ où $\widehat{Q} = \mathcal{P}(Q)$ et $\widehat{\delta}$ est définie par :

$$\forall P \subset Q, \forall a \in \Sigma, \widehat{\delta}(P, a) = \{\delta(p, a), p \in P\}$$

9. Montrer que l'existence d'un mot synchronisant pour M se ramène à un problème d'accessibilité de certain(s) état(s) depuis certain(s) état(s) de $\widehat{M}$.
10. En déduire que le langage $LS(M)$ des mots synchronisants de M est reconnaissable.
11. Déterminer puis représenter graphiquement un automate fini déterministe (pas nécessairement complet) reconnaissant $LS(M_0)$.
12. Montrer que si l'on sait résoudre le problème de l'existence d'un mot synchronisant, on sait dire, pour une machine M et un état q_0 de M choisi, s'il existe un mot u tel que pour tout état q de Q, le chemin menant de q à $\delta^*(q, u)$ passe forcément par q_0.

1.2 Réduction depuis SAT

On cherche à montrer que le problème de décision `Mot synchronisant` :

- ⋆ **Instance** : une machine M et un entier m.
- ⋆ **Question** : M possède-t-elle un mot synchronisant de taille $\leq m$?

est un problème au moins aussi difficile que le problème `SAT`. On s'intéresse dans cette partie à la satisfiabilité d'une formule logique sous forme normale conjonctive. Par exemple,

$$\varphi_0 = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee x_4)$$

est une formule sous forme normale conjonctive formée de trois clauses et portant sur un ensemble de 4 variables $\mathcal{V}_0 = \{x_1, x_2, x_3, x_4\}$.

Soit φ une formule sous forme normale conjonctive, composée de n clauses et faisant intervenir m variables d'un ensemble $\mathcal{V}$. On suppose les clauses numérotées c_1, c_2, …, c_n. On veut ramener le problème de la satisfiabilité d'une telle formule au problème de la recherche d'un mot synchronisant de longueur inférieure ou égale à m sur une certaine machine.

On introduit pour cela la machine $M_\varphi = (Q, \Sigma, \delta)$ associée à φ par :

— Q est formé de $mn + n + 1$ états, à savoir un état particulier noté f et $n(m+1)$ autres états qu'on notera $q_{i,j}$ avec $(i,j) \in [\![1,n]\!] \times [\![1, m+1]\!]$;

— $\Sigma = \{0, 1\}$;

— δ est définie par :

  — f est un états puits, c'est-à-dire que $\delta(f,0) = \delta(f,1) = f$;

  — pour tout $i \in [\![1,n]\!]$, $\delta(q_{i,m+1}, 0) = \delta(q_{i,m+1}, 1) = f$;

  — pour tout $i \in [\![1,n]\!]$ et $j \in [\![1,m]\!]$,

$$\delta(q_{i,j}, 0) = \begin{cases} f & \text{si le littéral } \overline{x_j} \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

$$\delta(q_{i,j}, 1) = \begin{cases} f & \text{si le littéral } x_j \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

13. Représenter graphiquement M_{φ_0} où φ_0 est la formule donnée précédemment.
14. Donner un modèle $\mu \in \{0,1\}^{\mathcal{V}_0}$ de φ_0. Le mot $\mu(x_1)\mu(x_2)\mu(x_3)\mu(x_4)$ est-il synchronisant ?
15. Montrer que tout mot u de longueur $m+1$ est synchronisant. À quelle condition sur les $\delta^*(q_{i,1}, u)$ un mot u de longueur m est-il synchronisant ?
16. Montrer que si une formule φ est satisfiable, tout modèle de φ donne un mot de longueur m synchronisant pour M_φ. On détaillera la construction de ce mot.
17. Réciproquement, montrer que si M_φ possède un mot synchronisant de longueur inférieure ou égale à m, alors φ est satisfiable. Décrire comment construire un modèle de φ.
18. Montrer que le problème Mot synchronisant est NP difficile.

    On va maintenant montrer que ce problème appartient à la classe NP.
19. Fixons deux états q et q' de l'automate et supposons qu'il existe un mot $u_{q,q'}$ tel que $q.u_{q,q'} = q'.u_{q,q'}$, autrement dit ce mot synchronise les deux états. Montrer qu'on peut supposer sans perte de généralité que ce mot est de longueur au plus $\binom{n}{2}$.
20. En déduire que si un automate admet un mot synchronisant alors il en admet un de longueur $O(n^3)$.
21. On propose la preuve suivante pour montrer que Mot Synchronisant est dans NP :

    On prend pour ensemble de certificats l'ensemble des mots de Σ^*. Pour une instance positive de Mot Synchronisant, on donne comme certificat un mot synchronisant de longueur inférieure ou égale à m. La fonction de vérification teste si le mot est bien de taille au plus m et qu'il est bien synchronisant comme traité dans la question 7 de la première partie. La fonction de vérification est bien en temps polynomial.

    Expliquer pourquoi cette preuve est incomplète et la compléter.
22. Montrer que le problème Mot Synchronisant est NP-complet.

2 COMPLEXITÉ ET APPROXIMATION : LE PROBLÈME DU SAC À DOS (EN OCAML)

On rappelle que le problème SUBSETSUM est défini comme suit :

Instance : $x_0, \dots, x_{n-1} \in \mathbb{N}^*$ et $S \in \mathbb{N}$

Question : existe-t-il $I \subseteq [0 \dots n-1]$ tel que $\sum_{i \in I} x_i = S$?

On appelle *objet* un couple (v, w) d'entiers naturels non nuls, où v est la *valeur* de l'objet et w son *poids*.
On définit alors les deux problèmes de décision suivants :

— 0/1-KNAPSACK

Instance : n objets (v_i, w_i) et deux entiers naturels V et W

Question : existe-t-il $I \subseteq [0 \dots n-1]$ tel que $\sum_{i \in I} v_i \geq V$ et $\sum_{i \in I} w_i \leq W$?

— KNAPSACK

Instance : n objets (v_i, w_i) et deux entiers naturels V et W

Question : existe-t-il $x_0, \dots, x_{n-1} \in \mathbb{N}$ tels que $\sum_{i=0}^{n-1} x_i v_i \geq V$ et $\sum_{i=0}^{n-1} x_i w_i \leq W$?

2.1 RÉDUCTIONS

On admet que le problème SUBSETSUM est NP-complet.

1. Montrer que les problèmes KNAPSACK et 0/1-KNAPSACK sont dans NP.
2. En réduisant SUBSETSUM, montrer que 0/1-KNAPSACK est NP-complet.

   On cherche désormais à montrer que KNAPSACK est NP-complet. Pour cela, on considère une instance $F = (X, S)$, avec $X = x_0, \dots, x_{n-1}$ de SUBSETSUM et l'on définit :
   — $B = 1 + \max(x_0, \dots, x_{n-1}, S)$;
   — $y_i = (2^n + 2^i)nB$ pour $0 \leq i < n$;
   — $y'_i = y_i + x_i$ pour $0 \leq i < n$;
   — $S' = \left(n2^n + \sum_{i=0}^{n-1} 2^i\right) nB + S$.

   On considère alors l'instance G de KNAPSACK où :
   — $V = S'$
   — $W = S'$
   — les $2n$ objets sont les couples (y_i, y_i) et (y'_i, y'_i) pour $0 \leq i < n$. *Chaque objet a donc une valeur égale à son poids.*
3. On suppose dans cette question que F est une instance positive de SUBSETSUM. Montrer que G est une instance positive de KNAPSACK.

   On suppose à présent que $\sum_{i=0}^{n-1} \lambda_i y_i + \sum_{i=0}^{n-1} \lambda'_i y'_i = S'$, avec les λ_i et λ'_i dans $\mathbb{N}$. On note $N = \sum_{i=0}^{n-1} (\lambda_i + \lambda'_i)$.
4. Montrer que $S' \geq 2^n nBN$ puis que $S' < 2^n n(n+1)B$.
   En déduire que que $N \leq n$.
5. Montrer qu'on a les deux égalités suivantes (on pourra utiliser l'unicité du quotient et du reste dans la division euclidienne de S' par nB) :
   — $S = \sum_{i=0}^{n-1} \lambda'_i x_i$
   — $N2^n + \sum_{i=0}^{n-1} (\lambda_i + \lambda'_i) 2^i = n2^n + \sum_{i=0}^{n-1} 2^i$

   On admet le résultat suivant :
   si $\sum_{i=0}^{n-1} \alpha_i 2^i \equiv \sum_{i=0}^{n-1} 2^i \mod 2^n$ et si $\sum_{i=0}^{n-1} \alpha_i \leq n$, alors $\alpha_0 = \dots = \alpha_{n-1} = 1$.
6. Montrer que KNAPSACK est NP-complet.

2.2 Algorithmes d'approximation

Dans cette partie et la suivante, on s'intéresse à la variante « optimisation » des problèmes Knapsack et 0/1-Knapsack :

— 0/1-Knapsack$_o$

Instance : n objets (v_i, w_i) et un entier W

Solution : une partie $I \subseteq [0 \dots n-1]$ telle que $\sum_{i\in I} w_i \leq W$

But : maximiser $\sum_{i\in I} v_i$

— Knapsack$_o$

Instance : n objets (v_i, w_i) et un entier W

Solution : n entiers naturels $\lambda_0, \dots, \lambda_{n-1}$ tels que $\sum_{i=0}^{n-1} \lambda_i w_i \leq W$

But : maximiser $\sum_{i=0}^{n-1} \lambda_i v_i$

Dans toute la suite, on suppose que tous les w_i sont inférieurs ou égaux à W (sans perte de généralité, puisque les objets ne vérifiant pas cette condition sont inutiles et peuvent facilement être éliminés).

On considère l'algorithme glouton suivant :

— on trie les objets par valeur décroissante de v_i/w_i ;

— on considère les objets dans cet ordre, et l'on prend chaque objet autant de fois que possible (étant donné la capacité qui reste disponible).

Par exemple, pour l'instance $o_0 = (10, 4)$, $o_1 = (20, 6)$, $o_2 = (3, 3)$ et $W = 17$:

— on considère o_1 en premier et l'on fixe $\lambda_1 = 2$;

— on considère ensuite o_0 et l'on fixe $\lambda_0 = 1$;

— on considère finalement o_2 et l'on fixe $\lambda_2 = 0$.

Début algorithme
- Trier les objets par v_i/w_i décroissant.
- $\lambda \leftarrow (0, \dots, 0)$ (taille n)
- $R \leftarrow W$
- **Pour** $i = 0$ à $n-1$ **Faire**
  - Poser $\lambda[i]$ maximal tel que $\lambda[i] \cdot w_i \leq R$
  - $R \leftarrow R - \lambda[i] \cdot w_i$
- **Renvoyer** λ

Algorithme 1 : Algorithme glouton pour Knapsack$_o$

7. On note v^* la valeur totale d'une solution optimale pour une instance $(v_0, \dots, v_{n-1}), (w_0, \dots, w_{n-1}), W$ (que l'on suppose triée par v_i/w_i décroissants). Montrer que $v^* \leq \frac{v_0}{w_0} W$.

8. Montrer que l'algorithme 1 (au verso) fournit une $\frac{1}{2}$-approximation pour Knapsack (on pourra commencer par justifier que $2\lambda_0 w_0 \geq (1 + \lambda_0) w_0 > W$).
   On considère une instance de Knapsack donnée par deux tableaux `v` et `w` de même longueur n et un entier `capacity`.

   On suppose disposer d'une fonction `by_ratio : int array -> int array -> int array` qui prend en entrée les tableaux `v` et `w` et renvoie un tableau `indices` de taille n tel que :

   — les `indices` sont exactement les entiers de 0 à $n-1$;

   — si `i < j`, alors $\dfrac{v.(indices.(i))}{w.(indices.(i))} \geq \dfrac{v.(indices.(j))}{w.(indices.(j))}$.

9. Écrire une fonction OCaml `greedy` telle que l'appel `greedy v w capacity` renvoie le tableau `lambda` obtenu par l'algorithme ci-dessus. Les tableaux `v` et `w` seront supposés de même longueur et correspondent respectivement aux valeurs et poids des objets ; l'entier `capacity` correspond à W.

```
val greedy : int array -> int array -> int -> int array
```

10. Déterminer la complexité de `greedy` en fonction du nombre n d'objets.
11. En considérant l'instance avec deux objets $(1,1)$ et $(N-1,N)$ et $W=N$. Montrer qu'en appliquant le même principe pour 0/1-KNAPSACK$_o$, on n'obtient pas une α-approximation (quel que soit $\alpha > 0$).

    On considère la *relaxation* FRACTIONAL-0/1-KNAPSACK$_o$ du problème 0/1-KNAPSACK$_o$ dans laquelle on s'autorise à prendre une quantité fractionnelle de chaque objet :

    Instance : n objets (v_i, w_i) et un entier W

    Solution : n **rationnels** $\lambda_0, \ldots, \lambda_{n-1} \in [0,1]$ tels que $\sum_{i=0}^{n-1} \lambda_i w_i \leq W$

    But : maximiser $\sum_{i=0}^{n-1} \lambda_i v_i$
12. On considère un ensemble $(v_i, w_i)_{0 \leq i < n}$ d'objets, et l'on note V^* la valeur optimale associée pour 0/1-KNAPSACK$_o$ et V_f^* la valeur optimale associée pour FRACTIONAL-0/1-KNAPSACK$_o$. Montrer que $V^* \leq V_f^*$.
13. Proposer un algorithme glouton très simple résolvant FRACTIONAL-0/1-KNAPSACK$_o$ de manière optimale (ce que l'on justifiera très rapidement), en temps $O(n \log n)$.