MINISTÈRE
DE L'ÉDUCATION
NATIONALE
ET DE LA JEUNESSE
Liberté
Égalité
Fraternité

EAE INF 3

SESSION 2023

AGREGATION
CONCOURS EXTERNE

Section : INFORMATIQUE

ÉPREUVE SPÉCIFIQUE SELON L'OPTION CHOISIE :
- **ÉTUDE DE CAS INFORMATIQUE**
- **FONDEMENTS DE L'INFORMATIQUE**

Durée : 6 heures

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire.

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

► **Etude de cas informatique** :

Concours	Section/option	Epreuve	Matière
EAE	6200A	103	9424

► **Fondement de l'informatique** :

Concours	Section/option	Epreuve	Matière
EAE	6200A	103	9425

Épreuve spécifique

IMPRIMERIE NATIONALE – 23 0105 – D'après documents fournis

Étude de cas informatique

Préliminaires

L'énoncé proposé s'inspire d'un projet concernant des livraisons par des véhicules. Chaque partie comprend la définition d'un certain nombre de problématiques à résoudre et présente des objectifs concrets, ainsi que la réflexion sur les moyens de les atteindre.

Attendus. Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction.

Dépendances. Ce sujet contient cinq parties. Les différentes parties et un grand nombre de leurs questions sont largement indépendantes. Il est possible d'aborder les différentes parties dans l'ordre qui vous conviendra le mieux mais en indiquant clairement quelle question est répondue.

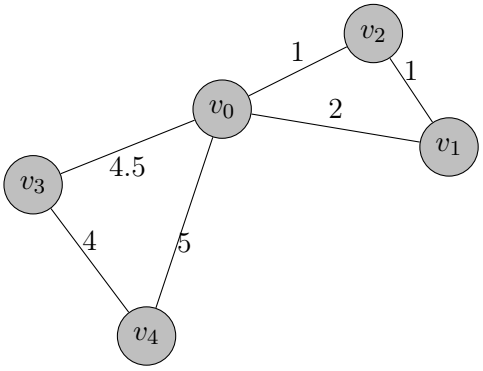
Partie I. Déterminer les tournées de véhicules

Dans cette partie, nous allons nous intéresser au problème de tournée de véhicules : un entrepôt stocke des produits qui doivent être livrés à des clients. Chaque client commande un nombre de produits (qui peuvent être différents). Les produits sont livrés par véhicule, par exemple un camion, et le but est de trouver des tournées de livraison qui minimisent la distance totale de kilomètres parcourus par les véhicules.

Ce problème est NP-complet (avec un véhicule seulement, c'est déjà le problème du voyageur de commerce) et nous allons donc nous intéresser à des heuristiques.

Nous allons utiliser la représentation via un graphe non-orienté $G = (V, E)$ avec les sommets représentant les différentes implantations des clients et les arêtes représentent les chemins entre les clients. L'entrepôt se situe au sommet v_0 et les n sommets restants représentent les clients. On considère que tous les sommets sont accessibles à partir de v_0 (donc le graphe est connexe). De plus, les arêtes e_{ij} sont pondérées par d_{ij}, la distance entre v_i et v_j. Le nombre maximum de véhicules est n, ce qui serait la situation dans laquelle exactement un véhicule est associé à chaque destination de livraison (client). Chaque véhicule a une capacité maximale C de produits qu'il peut charger en nombre de palettes. Pour finir, chaque client v_i, $i \geq 1$, a une demande w_i de produits. Un exemple est donné dans la figure 1 où on voit le graphe d'un réseau routier (figure 1(a) avec l'entrepôt au milieu et 4 clients) et sa représentation sous forme de matrice d'adjacence (figure 1(c)) ainsi que les demandes des clients (table 1(b) dans la figure 1).

Pour commencer, nous allons calculer les plus courtes distances entre tous les sommets pour un graphe donné. Ceci peut être fait avec l'**algorithme de Floyd-Warshall**. L'algorithme calcule, pour chaque paire de sommets, la distance minimale parmi tous les chemins entre ces



(a) Carte

client	v_1	v_2	v_3	v_4
demande	5	6	4	3

(b) Demandes en nombre de palettes

$$A = \begin{pmatrix} 0 & 2 & 1 & 4{,}5 & 5 \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 1 & 0 & \infty & \infty \\ 4{,}5 & \infty & \infty & 0 & 4 \\ 5 & \infty & \infty & 4 & 0 \end{pmatrix}$$

(c) Matrice d'adjacence

$$dist = \begin{pmatrix} 0 & 2 & 1 & 4{,}5 & 5 \\ 2 & 0 & 1 & 6{,}5 & 7 \\ 1 & 1 & 0 & 5{,}5 & 6 \\ 4{,}5 & 6{,}5 & 5{,}5 & 0 & 4 \\ 5 & 7 & 6 & 4 & 0 \end{pmatrix}$$

(d) Matrice des distances

FIGURE 1 – Exemple d'un graphe et sa représentation sous forme de matrice d'adjacence et sous forme de matrice des distances.

deux sommets. Le pseudo-code est donné dans l'algorithme 1. La matrice des distances produit par l'algorithme 1 en prenant la matrice d'adjacence de la figure 1(c)) en entrée, est donnée dans la figure 1(d)).

Question 1. Proposer une implémentation en Python de l'algorithme Floyd-Warshall présenté dans l'algorithme 1. La fonction prend la matrice A d'adjacence pondérée d'un réseau routier et produit une matrice *dist* des plus courtes distances entre tous les sommets. On considère pouvoir utiliser `np.inf` comme infini.

$dist \leftarrow A$
foreach vertex $z \in V$ **do**
  foreach vertex $x \in V$ **do**
    foreach vertex $y \in V$ **do**
      if $dist(x,z) \neq \infty$ & $dist(z,y) \neq \infty$ & $dist(x,z) + dist(z,y) < dist(x,y)$ **then**
        $dist(x,y) \leftarrow dist(x,z) + dist(z,y)$
return *dist*

Algorithme 1 : Floyd-Warshall.

Maintenant que nous savons créer la matrice des plus courtes distances à partir d'une matrice d'adjacence, dans la suite du sujet nous allons considérer directement des matrices de distances en entrée des algorithmes.

Nous allons aborder le problème des tournées des véhicules. Nous supposons que la demande de chaque client est inférieure à la capacité des véhicules, c'est-à-dire un client peut être livré en une fois par un véhicule.

Une première approche pour créer des tournées de véhicules peut être la suivante (**algorithme naïf**) : étant donnée la matrice des distances les plus courtes entre tous les points de la carte produit par l'algorithme Floyd-Warshall, on prend un premier véhicule et on lui affecte le client le plus proche. Puis, depuis ce client, on rajoute le client le plus proche de lui qui n'est pas encore affecté à la tournée si la capacité de chargement du véhicule n'est pas dépassée par cette affectation. On continue ainsi jusqu'à ce que la capacité restante du véhicule soit trop petite pour livrer en entier le client qu'on souhaite rajouter à la tournée. Dans ce cas, on commence une nouvelle tournée avec un autre véhicule selon le même principe jusqu'à ce que tous les clients soient affectés à des tournées. Chaque véhicule rentre à l'entrepôt à la fin de sa tournée.

Question 2. On considère la carte de la figure 1(a) et sa matrice des plus courtes distances (figure 1(d)). Chaque véhicule peut charger C=12 palettes. Les demandes de palettes des clients sont données dans la table 1(b) de la figure 1.

1. Appliquer l'**algorithme naïf** expliqué au dessus et donner les tournées avec leurs coûts en termes de nombre de kilomètres pour l'ensemble des véhicules. Le coût d'une tournée inclut le retour du véhicule à l'entrepôt via le plus court chemin.
2. En considérant l'objectif de minimiser la distance totale des kilomètres parcourus par tous les véhicules, peut-on trouver une meilleure solution ? Si oui, quelle est-elle ?

Question 3. Est-ce que la suppression du client v_2 change quelque chose pour la répartition des clients restants en tournées ? Expliquer.

Question 4. Implémenter en Python l'**algorithme naïf** via la fonction `AlgoNaif(dist, demandes, C)`. L'algorithme prend en entrée une matrice de distances entre tous les clients et l'entrepôt, les demandes des clients sous forme d'une liste ainsi que la capacité C (constante entière) des véhicules à disposition.

Question 5. Il est possible de trouver des situations dans lesquelles l'algorithme n'est pas optimal concernant notre critère de la distance totale des tournées. Donner deux configurations pour lesquelles l'algorithme se fait piéger. Un ou plusieurs dessins sont attendus.

Nous allons maintenant regarder une approche plus sophistiquée, l'**algorithme de Clarke et Wright** (*Savings algorithm*), dont le principe est le suivant :

1. On commence avec n tournées : $v_0 \to v_i \to v_0$, pour tout $i \geq 1$, c'est-à-dire chaque client est livré par un véhicule différent.
2. Calculer ensuite les économies $s(i, j)$ (*savings*) pour fusionner deux clients v_i et v_j en une même tournée : $s(i, j) = dist(i, 0) + dist(0, j) - dist(i, j)$, pour tout $i, j \geq 1$ et $i \neq j$;
3. Trier les économies par ordre décroissant ;

Tournez la page S.V.P.

4. Commencer en tête de la liste (restante) des économies, fusionner les deux tournées associées avec l'économie (restante) la plus élevée, si :

   (a) Les deux clients ne sont pas déjà dans la même tournée ;

   (b) Aucun des deux clients n'est à l'intérieur de sa tournée : Les deux clients sont connectés directement à l'entrepôt dans leur tournée respective et donc sont livrés en premier ou en dernier dans leur tournée respective ;

   (c) La somme des demandes des deux tournées à fusionner ne dépasse pas la capacité maximum du véhicule.

5. Répéter l'étape 4 jusqu'à ce que plus aucune économie ne puisse être faite.

Nous allons utiliser une classe `Tournee` qui représente la tournée d'un véhicule et qui propose les méthodes suivantes :

- `distance(self, distance)` : qui renvoie la longueur actuelle de la tournée,
- `fusionnable(self, autreTournee, clients, capacite)` : teste si la tournée peut être fusionnée avec `autreTournee`. Elle teste notamment les différents cas du point (2) de l'**algorithme de Clarke et Wright**.
- `fusion(self, autreTournee, clients)` : qui prend en entrée la tournée avec laquelle elle doit faire la fusion et une liste de clients par lesquels la tournée fusionnée doit être reliée.
- `affiche(self)` : affiche la tournée avec la liste des différents clients à parcourir et le chargement total de cette tournée.

Le listing 1 donne des exemples d'appel.

Listing 1 – Code d'utilisation de la classe `Tournee`.

```
tournee1.affiche() # tournee 0 2 3 4 0 chargement 8
tournee2 = Tournee(1,3) #client 1 demande 3 palettes
tournee1.fusionnable(tournee2, [3,1], 12) #False
tournee1.fusionnable(tournee2, [2,1], 12) #True
tournee1.fusion(tournee2, [2,1])
tournee1.affiche() # tournee 0 4 3 2 1 0, chargement 11
# le parcours 0-4-3-2-1-0 est equivalent a 0-1-2-3-4-0
```

Question 6. Implémenter la classe `Tournee` en Python.

Question 7. En utilisant la classe `Tournee`, implémenter en Python l'**algorithme de Clarke et Wright** via la fonction `ClarkeWright(dist, demandes, C)`. L'algorithme prend en entrée la matrice des distances des plus courts chemins, les demandes des clients et la capacité des véhicules. Il renvoie la liste des tournées ainsi que la longueur totale des tournées.

Question 8. Il peut être nécessaire de livrer certaines clients en urgence. Discuter (sans faire l'implémentation dans le code) des modifications de l'algorithme précédent nécessaires pour prendre en compte des priorités dans la liste des clients.

La version actuelle du problème de tournée de véhicules fait abstraction d'un certain nombre de contraintes supplémentaires qui font partie du cas réel, telles que le temps de conduite des conducteurs, la date de contrôle technique des véhicules, leur capacité ou leur plaque d'immatriculation.

On veut maintenant tenir compte de la différence de caractéristiques des différents véhicules. On considère avoir une classe `Vehicule` dont une instance sera associée à chaque tournée. Un véhicule sera associé à une tournée lors de la création de cette dernière. Dans les trois questions suivantes, on se limitera à considérer qu'un véhicule ne possède comme attributs que sa capacité en nombre de palettes (un entier) et une plaque d'immatriculation (une chaîne de caractères). On veut modifier la classe `Tournee` pour intégrer les véhicules, sans modifier la liste des méthodes de cette classe.

Question 9. Est-il nécessaire de modifier les arguments des méthodes de `Tournee` ? Si oui, donner et expliciter les prototypes des méthodes ainsi modifiées ou ajoutées, *i.e.* donnez leur nom, leurs arguments, ainsi que la sémantique des arguments ajoutés, enlevés ou modifiés.

Question 10. Est-il nécessaire de modifier le code des méthodes de `Tournee` ? Si oui, lesquelles (sans donner le code ainsi modifié) ?

Certaines informations concernant les véhicules (la plaque d'immatriculation par exemple) sont utiles pour l'exécution des tournées, mais ne sont pas utile dans les algorithmes précédents. Pour que l'équipe de développement en charge de programmer `Tournee` et celle en charge de programmer `Vehicule` travaillent efficacement, on définit une classe `Transport` qui se limite à garantir l'accès à la capacité. Les informations plus précises sur chaque véhicule resteront quand à elles dans `Vehicule`.

Question 11. En se limitant à la capacité (`capacite`) et au numéro d'immatriculation (`immatriculation`), proposer les classes `Transport` et `Vehicule` en vous limitant à leur définition (ligne commençant par `class`) et à leur constructeur.

Partie II. Prise en compte du réseau

Lors de la tournée elle-même, les véhicules sont en communication constante avec le service central de coordination de la flotte. Cette communication réseau utilise des applications présentes dans chaque véhicule qui envoient régulièrement des données (vitesse, position...) au service central. Les communications entre les véhicules et le serveur peuvent être compliquées : zones de non couverture, tunnels, conditions météo...

Question 12. On suppose le cas où un véhicule se trouve bloqué dans un tunnel empêchant la communication. Que se passe-t-il (couche transport du point de vue de l'application embarquée) si on suppose que le véhicule veut envoyer une information au serveur en tentant d'établir une communication basée sur TCP ?

Tournez la page S.V.P.

Un blocage long dans une zone non connectée peut être lié à un embouteillage. Dans ce cas-là, la densité de véhicules peut permettre de transmettre des informations en utilisant un principe de communication inter-véhicules. On suppose pour les questions suivantes que tous les véhicules sont équipés d'un système de communication local sans fil compatible.

On teste un premier protocole de routage. Quand le véhicule veut envoyer une donnée au serveur, il l'envoie (`broadcast`) à tous les véhicules autour de lui. Ces derniers font de même sauf s'ils sont capables de contacter l'extérieur (véhicule de *bordure*). Dans ce cas-là, ils envoient le message au serveur.

Question 13. Ce protocole a un défaut fondamental, quel est-il ? Comment le corriger ?

Une fois ce problème corrigé, on se pose la question de la réception au niveau du serveur. On suppose avoir utilisé le protocole TCP entre les véhicules de *bordure* et le serveur. Si plusieurs véhicules sont en bordure, plusieurs messages applicatifs vont donc être envoyés vers le serveur pour la même donnée.

Question 14. Est-il nécessaire, au niveau de l'application recevant les données sur le serveur, de gérer ces multiples messages contenant des données identiques, ou est-ce déjà géré au niveau TCP ? Justifier votre réponse.

Lorsque la densité de véhicules est forte (comme lors d'un embouteillage), la distance entre les véhicules est faible par rapport à la portée de la communication sans fil. Lors d'un embouteillage on peut avoir plusieurs dizaines de véhicules à portée de communication directe. Dans un premier temps, on considère qu'un seul véhicule tente d'envoyer un message vers le serveur extérieur.

Question 15. Dans le cadre où un seul véhicule veut envoyer un message, tous les messages intermédiaires entre le véhicule initial et les véhicules de *bordure* sont-ils nécessaires ? Justifier.

Question 16. Lors d'une communication sans fil, il est possible d'obtenir au niveau applicatif la qualité du signal portant un message. Plus la distance entre l'émetteur et le récepteur est grande, plus le message est faible. Proposez un algorithme de retransmission de message permettant d'optimiser le nombre de messages utilisés.

Question 17. Discuter l'impact de cet algorithme en terme de nombre de messages et de latence. On pourra fixer par exemple le nombre de véhicule à distance de communication sans fil comme étant une constante.

Après des tests dans les situations d'embouteillages dans des tunnels, on réalise que plusieurs véhicules peuvent vouloir transmettre de l'information vers le serveur extérieur. Le nombre de messages émis pour être retransmis entre les véhicules est proportionnel au nombre de véhicules.

Le premier problème est lié à la nature des communications sans-fil. Si deux véhicules tentent de communiquer en même temps, les deux messages vont être brouillés.

Question 18. À quel couche du modèle OSI se pose ce premier problème ? Proposer une solution pour régler ce problème.

Le second problème est lié au nombre de messages de contenus différents.

Question 19. Tous les messages sont retransmis de proche en proche vers la bordure. Comment réduire le nombre de messages lorsqu'un grand nombre de véhicules tentent chacun d'envoyer indépendamment des données vers un serveur extérieur ?

Partie III. Archivage de la position, de la vitesse

Pour respecter la loi, les véhicules professionnels doivent conserver des informations légales, principalement la vitesse, dans un tachygraphe. On décide de mettre en place un tel tachygraphe numérique. On profite de sa mise en place pour ajouter une fonctionnalité. Les données sont envoyées au serveur en temps réel quand le véhicule est joignable, mais les données sont stockées localement dans le cas contraire. Pour garantir les aspects réglementaires (non modification des données par le personnel de l'entreprise), on utilise une machine virtuelle qui peut être déplacée entre le serveur et le véhicule lorsque la qualité de la communication entre ces deux éléments se dégrade. Chaque véhicule est suivi par une application tachygraphe qui se trouve dans une machine virtuelle spécifique à ce véhicule.

On considère que le serveur et l'ordinateur de bord sont d'architectures différentes (par exemple d'architecture x86 pour les serveurs et de type ARM pour les véhicules).

Question 20. Est-il possible de migrer l'application tachygraphe entre le serveur et le véhicule ? Si oui, expliciter l'impact sur les performances.

Cette application de tachygraphe permet de centraliser l'information pour plusieurs usages : archivage légal, mais aussi information du centre de coordination des véhicules, affichage pour le conducteur... Comme l'accès et la modification des données est complexe, elle peut prendre du temps. Aussi, il faut gérer en interne la synchronisation d'accès aux données pour plusieurs processus fournissant les services pour ces différents usages.

On suppose avoir deux fonctions `lecture(...)` et `ecriture(...)`, déjà implémentées, permettant respectivement de lire et d'écrire ces données qui peuvent prendre du temps. On veut avoir les propriétés suivantes :

1. Plusieurs lectures peuvent avoir lieu en même temps,
2. Les écritures sont exclusives (sans lecture ni écriture en même temps),
3. En cas de choix, priorité aux lecteurs.

Tournez la page S.V.P.

Un premier test est le suivant :

Listing 2 – code de la classe `Acces`.

```
import threading

class Acces:
    def __init__(self):
        self.mutex = threading.Semaphore(value = 1)

    def debut_lecture():
        self.mutex.acquire() # P
    def fin_lecture():
        self.mutex.release() # V

    def debut_ecriture():
        self.mutex.acquire()
    def fin_ecriture():
        self.mutex.release()
```

Avec une utilisation type (en considérant la variable `synchro` de classe `Acces`) :

Listing 3 – code d'utilisation de la classe `Acces`.

```
# Pour une lecture
...
synchro.debut_lecture()
valeur = lecture(...)
synchro.fin_lecture()
...

# Pour une ecriture
...
synchro.debut_ecriture()
ecriture(...)
synchro.fin_ecriture()
...
```

Question 21. Pour chacune des trois propriétés, expliquer pourquoi elle est respectée ou pourquoi elle ne l'est pas.

Question 22. Proposer (en Python) une classe `Acces_v2` permettant d'implémenter l'ensemble des propriétés souhaitées en conservant les mêmes méthodes.

Question 23. Donner un exemple de famine dans le cadre des propriétés souhaitées.

Partie IV. Gestion de tournées

Voici en figure 2 le schéma relationnel d'une base de données qui permet à l'entreprise de faire la gestion de ses tournées. La spécification des domaines est donnée dans le listing 4.

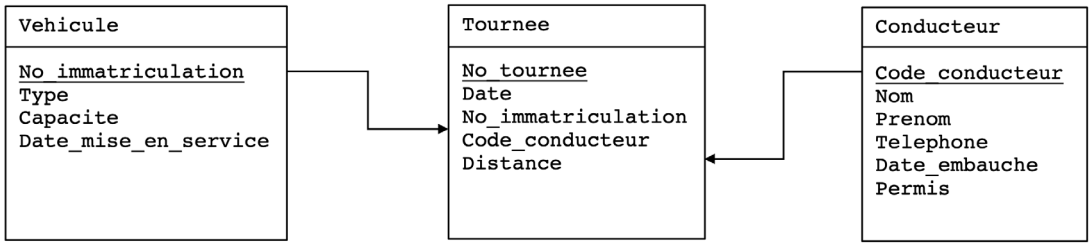


FIGURE 2 – Schéma relationnel de la base de donnée **Gestion de tournées**.

Listing 4 – Spécification de la base de données.

```
VEHICULE(*No_immatriculation(char(10)), Type(text), Capacite(float),
Date_mise_en_service(date))

TOURNEE(*No_tournee(int), Date(date), No_immatriculation(char(10)),
Code_conducteur(char(8)), Distance(float))

CONDUCTEUR(*Code_conducteur(char(8)), Nom(char(30)),
Prenom(char(40)), Telephone(texte), Date_embauche(date),
Permis(texte))
```

Question 24. Écrire une requête en SQL qui permet d'afficher combien de conducteurs ont un permis C1E (élément `Permis` dans la table précédente).

Question 25. Écrire une requête en SQL qui affiche la distance totale parcourue par John Doe en novembre 2021. On rappelle que les dates sont exprimées par défaut sous la forme `YYYY-MM-DD` dans les requêtes.

Question 26. Écrire une requête en SQL qui permet de calculer la capacité cumulée des véhicules selon le permis de leur conducteur.

Les différents véhicules nécessitent des permis de conduire différents. En effet, selon leur poids total autorisé en charge (PTAC), ou selon la présence éventuelle d'un attelage de remorque, il faut avoir le permis correspondant. On trouve un résumé des différents permis de transport de marchandises dans la table 1 et la figure 3. Le permis C1 permet ainsi de conduire un véhicule isolé dont le PTAC est compris entre 3,5t et 7,5t, tandis que le permis C1E permet de conduire un ensemble de véhicules dont le tracteur appartient à la catégorie C1 et dont le poids total ne dépasse pas 12t.

Dans sa forme actuelle, la base de données ne permet pas d'éviter les erreurs d'affectation. Il est pour l'instant tout à fait possible d'affecter un conducteur de moins de 21 ans à un véhicule isolé de plus de 7,5t de PTAC. Pour empêcher ce type d'erreur, l'entreprise souhaite améliorer la base de données en modifiant le schéma de manière à pouvoir effectuer une vérification

Tournez la page S.V.P.

Permis	PTAC	Véhicule isolé	Remorque	Âge minimum	Équivalence
C1	3,5 à 7,5 t	oui	< 750 kg	18	-
C1E	$> 3{,}5$ t, max 12t	non	> 750 kg	18	-
C	$> 3{,}5$ t	oui	< 750 kg	21	C1
CE	$> 3{,}5$ t	non	> 750 kg	21	C1E

TABLE 1 – Les permis poids lourd pour le transport des marchandises.

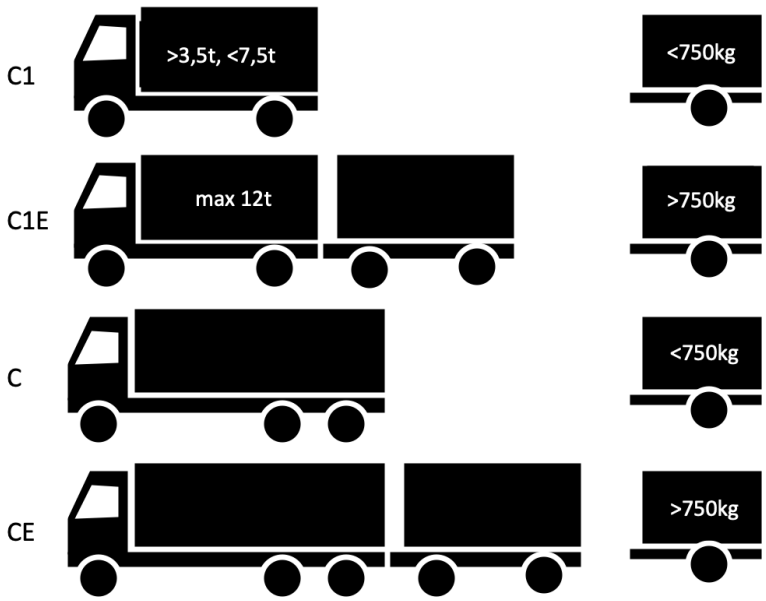


FIGURE 3 – Les permis poids lourd pour le transport de marchandises.

automatique de compatibilité d'affectation entre un véhicule et un conducteur. Pour ceci, il est nécessaire de pouvoir stocker les spécifications des différents permis dans la base et il a été décidé d'ajouter une table `Permis` au schéma relationnel.

Question 27. Modéliser la nouvelle table `Permis` sous une forme similaire au listing 4 en précisant le domaine pour chaque attribut. (Pas de requête en SQL.)

Question 28. Une analyse plus approfondie du schéma relationnel montre que le type d'un véhicule est stocké sous la forme d'une simple chaîne de caractères. Discuter les limites de ce choix de modélisation.

L'entreprise a fait un inventaire de sa flotte de véhicules et il s'est avéré que les véhicules peuvent être catégorisés selon les types suivants :

Type	longueur	largeur	hauteur	PTAC	capacité palettes 80x120
Semi-remorque	16,5 m	2,55 m	4 m	26 t	33
Porteur 19 t	10,70 m	2,55 m	4 m	19 t	20
Porteur 12 t	9,5 m	2,55 m	4 m	12 t	14
20 m^3	7 m	2,5 m	3,10 m	3,5 t	8

Question 29. Proposez une extension du schéma relationnel qui permet la vérification automatique du permis de conduire lors d'une affectation d'un conducteur à un véhicule. Présenter votre proposition sous une forme similaire à la figure 2 et au listing 4. Justifiez vos choix.

Partie V. Utilisation des éléments mobiles

Dans cette partie, nous allons mettre en place une application web qui permettra à un chauffeur de visualiser ses retards dans sa tournée, ainsi que de valider ses livraisons.

Un aperçu de l'application est donné dans la figure 4. Le chauffeur verra la liste des clients à livrer avec l'heure de livraison. Le code couleur suivant permettra de visualiser directement l'état de la livraison :

blanc Le créneau de la livraison est dans le futur.

gris clair Le créneau de la livraison est dépassé d'une heure au plus.

gris foncé Le créneau de livraison est largement dépassé (plus d'une heure).

noir La livraison a été effectuée.

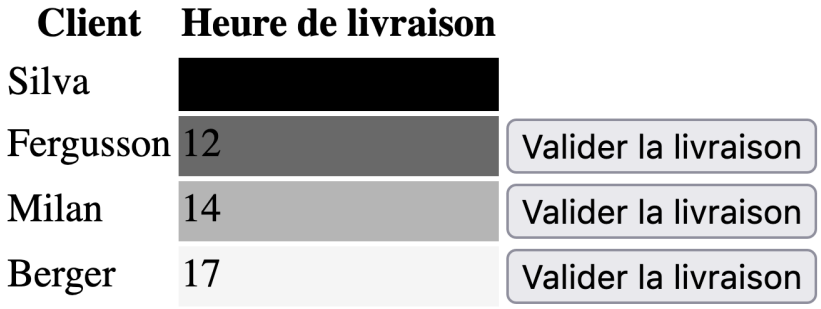


FIGURE 4 – Application web : Visualisation de tournée, heure actuelle : 14h13.

Le bouton **Valider la livraison** permet de changer le statut de la livraison et l'affichage du créneau horaire passe en noir.

Ainsi, on comprend dans l'exemple de la figure 4 que la livraison de Silva a été validée. La livraison pour Fergusson est très en retard tandis que la livraison pour Milan est un peu en retard. La livraison pour Berger peut être effectuée sans retard.

Question 30. En utilisant HTML et CSS uniquement, programmer la partie statique de la page, c'est-à-dire l'affichage du tableau sans la logique des changements de couleur en supposant qu'il n'y a que les quatre clients visibles sur la figure 4 à afficher.

Tournez la page S.V.P.

Listing 5 – Début du code de la page statique.

```
<!DOCTYPE html>
<html>
    <head>
        <style>
                .ontime {
                        background-color: white;
                }
                .hurry {
                        background-color:lightgrey;
                }
                .late {
                        background-color: dimgray;
                }
                .done {
                        background-color: black;
                }
        </style>
     </head>
     <body>
          <!-- TODO -->
     </body>
</html>
```

Question 31. En utilisant JavaScript, programmer l'affichage en couleurs des créneaux de livraison par rapport à l'heure actuelle. La première ligne du code est déjà écrite (voir listing 6).

Listing 6 – Début du code de la coloration automatique des créneaux de livraison.

```
  <script>
  let hours = document.querySelectorAll('.heure')

  const d = new Date()
  let heureActuelle = d.getHours()

  //TODO
  </script>
```

Question 32. En utilisant JavaScript, programmer la validation d'une livraison : dès que le bouton **Valider la livraison** a été actionné, le créneau de livraison s'affiche en noir et le bouton disparaît. La première ligne du code est déjà écrite (voir listing 7).

Listing 7 – Début du code de la fonctionnalité du bouton.

```
  <script>
  let rows = document.querySelectorAll(".row")
  //TODO
  </script>
```

Il arrive aux conducteurs de vouloir changer le créneau de livraison pour un client dans l'application. Dans ce cas, le conducteur saisit le nom du client ainsi que le nouveau créneau dans un formulaire avec deux champs dans l'application. L'application envoie une requête HTTP vers un serveur web qui s'occupe de vérifier si le changement de créneau est possible. Plus précisément, l'application envoie une requête GET vers /**traitement** avec les paramètres **client** et **nHeure** avec les données fournies par le conducteur via le formulaire web. Si on souhaite changer l'heure du client **Fergusson** à 18h et en supposant que l'adresse du serveur est `http://www.appli-tournee.fr`, alors un telle requête serait : `GET //www.appli-tournee.fr/traitement?client=Fergusson&nHeure=18` . Si la réponse du serveur est positive, alors l'affichage du créneau horaire du client doit changer sur le site. Par contre, en cas de réponse négative, une alerte sera donnée au conducteur signalant que le changement n'est pas possible. On rappelle que la fonction JavaScript "alert(message)" permet de lancer une alerte à l'utilisateur.

Question 33. En utilisant des fonctions asynchrones de JavaScript, programmer le script qui permet de changer le créneau horaire d'un client. Le listing 8 donne la première ligne du code.

Listing 8 – Début du code pour mettre à jour un créneau de livraison.

```
<script>
let buttonMAJ = document.getElementById("maj")
//TODO
</script>
```

Rappels

- `element.innerHTML` : la propriété `Element.innerHTML` de `Element` récupère ou définit la syntaxe HTML décrivant les descendants de l'élément.
- `EventTarget.addEventListener()` : la méthode `addEventListener()` d'un `EventTarget` attache une fonction à appeler chaque fois que l'évènement spécifié est envoyé à la cible.
- `document.querySelector` : la méthode `querySelector()` de l'interface `Document` renvoie le premier `Element` dans le document correspondant au sélecteur - ou groupe de sélecteurs - spécifié(s), ou `null` si aucune correspondance n'est trouvée.

Fondements de l'informatique

Préliminaires généraux Les questions de programmation doivent être traitées en langage *OCaml*. On pourra utiliser toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme `max`, `incr` ainsi que les opérateurs comme `@`). L'utilisation d'autres modules est interdite. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien satisfaites. On ne cherchera pas à gérer les exceptions. Lorsque les choix d'implémentation ne découlent pas directement des spécifications de l'énoncé, il est vivement conseillé de les expliquer.

Il est attendu des candidates et des candidats des réponses construites. Ils seront également évalués sur la précision, le soin et la clarté de la rédaction. Les questions avec une étoile (c'est-à-dire de la forme **Question***) sont a priori plus difficiles ou nécessitent plus d'attention. Il est autorisé d'admettre le résultat d'une question (en particulier celles avec une étoile) pour traiter les questions suivantes.

Ce sujet s'intéresse à différents modèles de neurones et réseaux de neurones formels, à leur puissance en tant que modèles de calculs, ainsi qu'à la complexité de leur apprentissage exact.

Fixons une fonction $\mathcal{A} : \mathbb{R} \to \mathbb{R}$ ($\mathbb{R}$ désigne l'ensemble des réels). Jusqu'à la fin de la partie IV, la fonction $\mathcal{A}$ (qui est appelée la *fonction d'activation*) sera la *fonction de Heaviside*, c'est-à-dire la fonction $\mathcal{H}(x)$ qui vaut 1 pour $x \geq 0$, 0 sinon.

Un *neurone à n entrées* est décrit par des paramètres $w_1, w_2, \ldots, w_n$ et h, qui sont des réels. Chacun des w_i est appelé un *poids*, et h est appelé le *seuil* du neurone. Un tel neurone calcule une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ de la façon suivante : sur les entrées $x_1, x_2, \ldots, x_n$, il prend la valeur $\mathcal{A}(w_1x_1 + w_2x_2 + \cdots + w_nx_n - h)$. Cette valeur s'appelle *sa valeur d'activation.*

Objectifs généraux du sujet Dans la partie I, on étudie les neurones pris isolément. De la même manière qu'en informatique en partant des portes logiques élémentaires comme le *ET*, le *OU* et la négation *NOT*, on s'intéresse classiquement aux circuits booléens construits à partir de ces portes comme modèles de calculs, on s'intéressera à partir de la partie II aux circuits ("réseaux") de neurones construits à partir de neurones.

Notez que les poids et les seuils sont a priori des réels quelconques. On parlera de neurone ou de réseau de neurones *à poids unitaires* lorsque tous les poids[1] prennent leur valeur dans $\{-1, 0, 1\}$.

1. Dans un réseau à poids unitaire, on n'impose rien sur les seuils : cela peut très bien être des réels quelconques.

Tournez la page S.V.P.

Partie I. Le cas d'un neurone à seuil

On note $\mathbf{B} = \{0, 1\}$: l'intuition dans tout le sujet est que le réel 0 représente la valeur logique *faux*, et que le réel 1 représente la valeur logique *vrai*. On répète que dans cette partie, comme jusqu'à la fin de la partie IV, la fonction d'activation $\mathcal{A}$ est la fonction de Heaviside : on parle de *neurone à seuil* dans ce cas.

1 Fonction booléennes linéaires à seuil

On dit qu'une fonction $F : \mathbf{B}^n \to \mathbf{B}$ est une *fonction booléenne linéaire à seuil* (et plus généralement qu'une fonction $F : S \subseteq \mathbb{R}^n \to \mathbf{B}$ est une *fonction linéaire à seuil*) si elle correspond à la fonction calculée par un neurone à seuil : dit autrement, il existe des réels $w_1, \ldots, w_n$ et h tels que, pour tout $\mathbf{x} = (x_1, \ldots x_n)$ dans $\mathbf{B}^n$ (ou plus généralement dans S), $F(\mathbf{x}) = 1$ si et seulement si $\sum_{i=1}^{n} w_i x_i \geq h$.

On appelle $(w_1, \ldots, w_n, h)$ une *représentation de* F. On dit que la représentation est à poids unitaires lorsque $w_1, w_2, \ldots, w_n$ restent dans $\{-1, 0, 1\}$. Bien entendu, une même fonction booléenne linéaire à seuil peut avoir plusieurs représentations.

Par exemple, la fonction *ET logique* à n-arguments $\mathrm{AND}_n(x_1, \ldots, x_n)$, qui vaut 1 si et seulement si chacun des x_i vaut 1, est une fonction booléenne linéaire à seuil (et même à poids unitaires). En effet, elle admet la représentation $(\underbrace{1, \ldots, 1}_{n}, n)$, car $\mathrm{AND}_n(x_1, \ldots, x_n) = 1$ si et seulement si $1x_1 + 1x_2 + \cdots + 1x_n \geq n$.

Question 1. Montrer que le *OU logique* (à n-arguments) OR_n, la *NEGATION logique* NOT_1 (à un argument) sont également des fonctions booléennes linéaires à seuil. Montrer que c'est également le cas de la fonction $\mathrm{MAJORITY}_n : \mathbf{B}^n \to \mathbf{B}$ définie comme la fonction qui vaut 1 si et seulement si au moins $n/2$ de ses entrées valent 1.

Question 2. Montrer que le *OU EXCLUSIF* à deux arguments (noté XOR_2 ou encore $\oplus$) n'est pas une fonction booléenne linéaire à seuil.

Question 3. En déduire que pour tout $n \geq 2$, la fonction parité à n arguments $\mathrm{PARITY}_n(x_1, \ldots, x_n)$ (qui vaut 1 si et seulement si un nombre pair de ses arguments valent 1) n'est pas une fonction booléenne linéaire à seuil.

2 Cas de domaines bornées

Lorsque δ est un nombre réel strictement positif, une représentation $(w_1, \ldots, w_n, h)$ est dite *δ-séparable* sur un ensemble $S \subseteq \mathbb{R}^n$ si pour tout $x_1, \ldots, x_n \in S$

$$\sum_{i=1}^{n} w_i x_i < h \text{ si et seulement si } \sum_{i=1}^{n} w_i x_i \leq h - \delta.$$

Une fonction linéaire à seuil F est dite séparable sur un ensemble S s'il existe $\delta > 0$ tel que F admette une représentation δ-séparable sur S.

Question 4. Montrer que toute fonction linéaire à seuil sur un ensemble fini est séparable.

La *masse* d'une représentation $(w_1, \ldots, w_n, h)$ est définie comme $\max\{|w_i| \mid 1 \leq i \leq n\}$, c'est-à-dire le maximum des valeurs absolues de ses poids.

Question 5. Montrer que pour tout $\delta > 0$, $S \subseteq \mathbb{R}^n$, et toute fonction linéaire à seuil F, si F possède une représentation qui est λ-séparable de masse w, alors F possède également une représentation δ-séparable de masse $w\delta/\lambda$.

On en déduit que pour tout $\delta > 0$, toute fonction linéaire à seuil F séparable possède une représentation δ-séparable.

Une fonction F est une fonction linéaire *à seuil nul* si c'est une fonction linéaire à seuil et qu'elle admet une représentation $(w_1, \ldots, w_n, 0)$, c'est-à-dire dont le seuil vaut 0.

Question 6. Montrer que pour toute fonction linéaire à seuil $F : \mathbb{R}^n \to \mathbf{B}$, on peut construire une fonction linéaire à seuil nul $G : \mathbb{R}^{n+1} \to \mathbf{B}$ (et donc avec une dimension (entrée) supplémentaire) telle que pour tout $x_1, \ldots, x_n \in \mathbf{B}$, $F(x_1, \ldots, x_n) = G(x_1, \ldots, x_n, 1)$.

Ne serait-ce que pour des raisons de représentation sur ordinateur, il est parfois intéressant de se ramener au cas où tous les poids sont entiers : on dit qu'une représentation $(w_1, \ldots, w_n, h)$ est entière *si chacun des w_i et h sont des éléments de $\mathbb{Z}$, l'ensemble des entiers relatifs.*

Question 7. Soit $\delta > 0$. On considère une fonction linéaire à seuil sur un ensemble borné $S \subseteq [0,1]^n$ avec une représentation n-séparable, de la forme $(w_1, \ldots, w_n, h)$ avec $w_1, \ldots, w_n \geq 0$ de masse nw/δ. Montrer que la fonction possède également une représentation entière de même dimension de masse au plus nw/δ.

Question 8. Montrer que toute fonction linéaire à seuil (sans restriction sur le signe des poids) sur un ensemble borné $S \subseteq [0,1]^n$ avec une représentation δ-séparable de masse w, possède une représentation entière de masse au plus nw/δ.

On déduit donc (de la question 8 et de la remarque qui suit la question 5) que ***toute fonction linéaire à seuil séparable sur un ensemble borné possède une représentation entière.***

3 Apprentissage d'un neurone à seuil

Soit $D = [0,1]$ ou $D = [-1,1]$. On considère dans cette partie *le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine D* : c'est-à-dire, étant donné un ensemble fini de N couples, c'est-à-dire $(\mathbf{x}_i, y_i)_{1 \leq i \leq N}$, avec $\mathbf{x}_i \in D^n, y_i \in \mathbf{B}$, on cherche à déterminer s'il existe une fonction linéaire à seuil F telle que pour tout $1 \leq i \leq N$, $F(\mathbf{x}_i) = y_i$. Si une telle fonction F existe, on dit que le problème d'apprentissage *admet une solution* (qui est la fonction F). On appelle n la dimension.

Tournez la page S.V.P.

L'ensemble $X = \{x_1, x_2, \ldots, x_N\}$ est appelé *l'ensemble des exemples*. Lorsque $y_i = 1$ on dit que $\mathbf{x}_i$ est un exemple *positif*. Lorsque $y_i = 0$ on dit que $\mathbf{x}_i$ est un exemple *négatif*.

On cherche à résoudre ce problème pour $D = [0, 1]$ dans le cas le plus général. Pour cela, on va chercher à le réduire à des variantes plus faciles à traiter. On parle de problème d'apprentissage :

— *à seuil nul* lorsqu'on cherche à déterminer s'il existe F, une fonction linéaire à seuil comme ci-dessus, avec de plus F à seuil nul.

— *à exemples négatifs* lorsque l'on a $y_i = 0$ pour tous les $1 \leq i \leq N$.

Question 9. Montrer que le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine D en dimension n se réduit au problème de l'apprentissage exact des fonctions linéaires à seuil nul sur le domaine D en dimension $n + 1$.

Question 10. Montrer que le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine $[0, 1]$ en dimension n se réduit au problème de l'apprentissage exact des fonctions linéaires à seuil nul à exemples négatifs sur le domaine $[-1, 1]$ en dimension $n + 1$.

Autrement dit, il suffit de savoir résoudre le problème de l'apprentissage exact dans le cas où F est à seuil nul, et où l'on n'a que des exemples négatifs, et $D = [-1, 1]$.

On va s'intéresser en fait au problème de calcul associé : étant donné X avec des exemples négatifs, $D = [-1, 1]$, lorsqu'il existe une fonction linéaire à seuil nul F solution, produire une représentation d'une telle fonction F. Introduisons pour cela l'algorithme suivant.

```
1  Procedure apprentissage(n, X)
2      for i ∈ {1, 2, ..., n} do
3          w_i := 0
4      repeat
5          fin := 1 ;
6          for x = (x_1, ..., x_n) ∈ X do
7              if Σ_{i=1}^{n} w_i x_i ≥ 0 then
8                  fin := 0 ;
9                  for i ∈ {1, 2, ..., n} do
10                     w_i := w_i − x_i
11     until fin = 1;
12     return (w_1, w_2, ..., w_n, 0)
```

Chaque itération de la boucle *repeat* est appelé *une époque*. Lorsque la condition $\sum_{i=1}^{n} w_i x_i \geq 0$ est satisfaite à la ligne 7, on dit qu'une *erreur à été commise*.

Question 11. On fixe la déclaration de type

```
type vecteur = float array
```

Proposer une implémentation `apprentissage: int -> vecteur array -> vecteur` de cet algorithme en *OCaml. On rappelle qu'en* OCaml*, les opérations arithmétiques sur les* `float` *sont désignées par +., -., *., /.*

On veut montrer que l'algorithme termine avec une représentation valide si et seulement si le problème d'apprentissage exact admet une solution à seuil nul.

Question 12. Montrer que si l'algorithme termine, alors le problème d'apprentissage exact à seuil nul défini par l'ensemble des exemples négatifs X admet une solution, dont une représentation est donnée par $(w_1, \ldots, w_n, 0)$.

Il reste à prouver l'implication réciproque.

Question 13. Soit $X \subseteq [-1, 1]^n$ un ensemble fini de N exemples négatifs, et F une fonction linéaire à seuil nul solution. Démontrer que si F possède une représentation $(v_1, \ldots, v_n, 0)$ qui est n-séparable alors l'algorithme termine et est correct.

On pourra considérer $d(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^{n} (w_i - v_i)^2$ où $\mathbf{w} = (w_1, \ldots, w_n)$ décrit les poids à chaque époque, et on rappelle que X est constitué d'exemples négatifs.

Dans le cas général, on considère F une fonction linéaire à seuil et un ensemble fini d'exemples. Par la question 4 et la partie 2, elle possède une représentation entière.

Question* 14. Soit $X \subseteq [-1, 1]^n$ un ensemble fini de N exemples négatifs. On suppose que le problème de l'apprentissage exact à exemples négatifs associé à X admet une solution 1-séparable à coefficients entiers et de masse inférieure ou égale à w. Montrer que l'algorithme commet alors au plus $\mathcal{O}\left(n^2 w^2\right)$ erreurs et termine en temps au plus $\mathcal{O}\left(n^2(n+N)w^2\right)$.

Partie II. D'un neurone à seuil à un réseau de neurones

On introduit maintenant les circuits de neurones, également appelés "réseaux de neurones".

Fixons un ensemble K. Fixons un ensemble $\mathcal{G}$ de *portes élémentaires* sur K : chaque porte élémentaire g de $\mathcal{G}$ est une fonction $g : K^{n_g} \to K$ pour un certain entier n_g, que l'on appelle son *arité*. Par exemple, on peut prendre $K = \mathbf{B}$, et $\mathcal{G} = \mathcal{G}_{bool} = \{\mathrm{AND}_2, \mathrm{OR}_2, \mathrm{NOT}_1, 0, 1\}$, où 0, et 1 sont d'arité 0, et l'arité des autres portes élémentaires est indiquée par l'indice dans son nom.

On définit la notion de circuits sur un ensemble de portes $\mathcal{G}$, en généralisant la notion usuelle de circuit booléen (qui correspond au cas $\mathcal{G} = \mathcal{G}_{bool}$ sur $K = \mathbf{B}$) à un ensemble de portes élémentaires $\mathcal{G}$ plus général.

Définition 1. *Un* circuit sur $\mathcal{G}$ *est donné par un graphe fini $(V \cup X, E)$ orienté sans cycle. Les sommets de X sont appelés des* entrées*, et n'ont pas d'arc entrant. Tout sommet de V est appelé* une porte*, et est étiqueté par un élément g de $\mathcal{G}$: si cet élément g est d'arité n_g, alors ce sommet possède exactement n_g arcs entrants. Les sommets de V qui n'ont pas d'arc sortant sont appelés des* sorties*.*

Tournez la page S.V.P.

Notons que les sommets de X, et de V qui ne sont pas des sorties, peuvent avoir plusieurs arcs sortants. S'il y a n entrées et m sorties, on fixe une numérotation de celles-ci de 1 à n et de 1 à m respectivement.

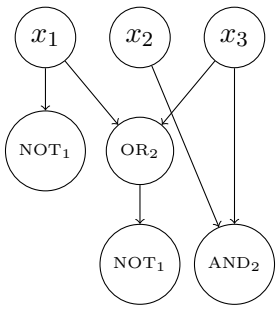
La *taille* d'un tel circuit est définie comme le nombre de sommets dans V, et donc son nombre de portes.

Étant donnée une valeur pour chacune des entrées, la valeur de chacune des sorties s'obtient en propageant la valeur des entrées vers les sorties en évaluant la valeur de chaque porte selon la fonction associée. On dit que le circuit calcule la fonction $f : S \subseteq K^n \to K^m$, si pour tout $(x_1, \ldots, x_n) \in S$, si on note $(y_1, \ldots, y_m) = f(x_1, \ldots, x_n) \in K^m$, alors si l'on fixe l'entrée numéro i à x_i, pour $1 \leq i \leq n$, la valeur de la sortie numéro j est y_j, pour $1 \leq j \leq m$.

Par exemple, un circuit booléen avec n entrées et m sorties calcule une fonction de $\mathbf{B}^n$ dans $\mathbf{B}^m$.

La fonction profondeur prof qui à un sommet de $V \cup X$ associe *sa profondeur* se définit inductivement de la façon suivante : $\text{prof}(x) = 0$ pour chaque $x \in X$, et $\text{prof}(v) = 1 + \max_{(s,v)\in E} \text{prof}(s)$. La *profondeur d'un circuit* est la plus grande profondeur d'un de ses sommets. On appelle *couche* k l'ensemble des sommets de profondeur k.

Voici un circuit booléen sur $\mathcal{G} = \mathcal{G}_{bool}$ à 3 entrées et 3 sorties, de taille 4. Si l'on numérote les sorties de gauche à droite, il calcule une certaine fonction $f : \mathbf{B}^3 \to \mathbf{B}^3$; on a par exemple $f(0, 1, 0) = (1, 1, 0)$.



Question 15. Quelle est la profondeur de ce circuit ? Détailler chacune de ses couches. Décrire complètement la fonction calculée par ce circuit.

4 Des circuits booléens aux réseaux de neurones à seuil

En prenant pour $\mathcal{G}$ l'ensemble des neurones à seuil (un neurone à n entrées ayant l'arité n) sur $K = \mathbb{R}$, on obtient un *circuit de neurones*, aussi appelé *un réseau de neurones*.

Cette construction se généralise au cas d'une fonction d'activation $\mathcal{A}$ différente de la fonction de Heaviside : nous reviendrons sur ce point à la fin de la partie IV.

Question 16. On souhaite coder les réseaux de neurones en *OCaml*. Pour la commodité d'implémentation en *OCaml*, on utilise une représentation différente (bien qu'équivalente) de celle donnée plus haut pour un réseau de neurones. *Ce choix de représentation en machine est fixé uniquement pour cette question et ne sera plus utilisé dans le reste du sujet.* On voit un réseau de neurones comme un graphe, où chaque arc est étiqueté par un poids, et chaque sommet est étiqueté par son seuil. On souhaite coder le graphe selon le principe d'une liste d'adjacence : supposons que les sommets sont numérotés de 0 à $n-1$. Le graphe est alors représenté sous la forme d'un tableau T de taille n tel que $T.(i)$ contienne la liste des extrémités et poids des arcs sortants du sommet i ainsi que le seuil associé au sommet i.

On définit les types suivants :

```
type sommet  = int
type seuil = float
type poids = float
type reseau_adjacence = ((sommet * poids) list * seuil) array
```

Proposer une implémentation en *OCaml* de la fonction `creer : int -> reseau_adjacence` qui permet de créer un réseau avec un nombre donné de sommets.

Même question pour `teste_arc : reseau_adjacence -> sommet -> sommet -> bool` qui permet de tester l'existence d'un arc entre deux sommets.

Même question pour la fonction

```
ajoute_arc : reseau_adjacence -> sommet -> sommet -> poids -> reseau_adjacence
```

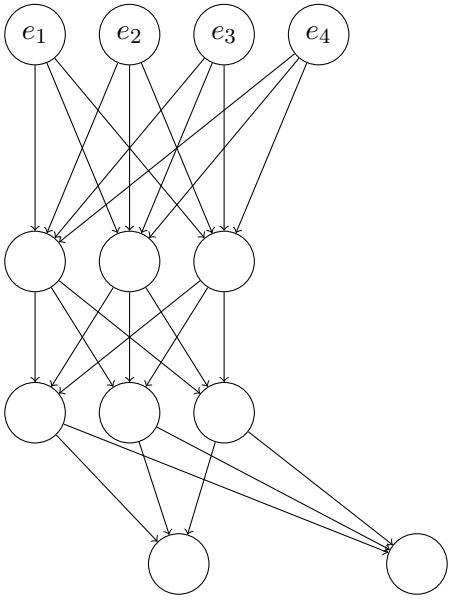
qui ajoute un arc entre deux sommets, avec un poids donné.

Question 17. Soit C un circuit booléen de taille t et de profondeur d qui calcule la fonction $f : \mathbf{B}^n \to \mathbf{B}^m$. Montrer qu'il existe un réseau de neurones à seuil à poids unitaires de taille t et de profondeur d qui calcule la fonction f.

On va voir dans la partie suivante qu'il est parfois possible de calculer certaines fonctions de façon beaucoup plus efficace en terme de taille avec des réseaux de neurones.

On dit qu'un réseau de neurones est *bien formé* si chaque arc (u, v) de E est tel que si u appartient à la couche i, alors v appartient à la couche $i+1$. Par exemple, un réseau qui aurait l'architecture (c'est-à-dire le graphe sous-jacent, les poids et seuils ne sont pas représentés) suivante est bien formé.

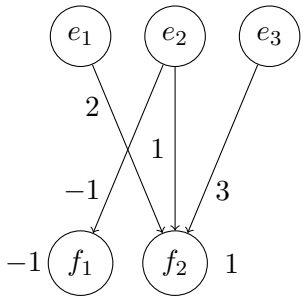
Tournez la page S.V.P.



Question 18. Montrer qu'étant donné un réseau N de neurones à seuil, on peut construire un réseau de neurones N' à seuil de même profondeur bien formé équivalent : si N possède n entrées, N' également, et N et N' calculent la même fonction.

Question 19. On utilise dans cette question une représentation en machine des réseaux de neurones différente de celle de la question 16 ; ici, on représente un réseau de neurones bien formé par une liste de couples l. Si n est le nombre de neurones de la $(i-1)$-ème couche (ou, pour $i = 0$, le nombre d'entrées du réseau) et m le nombre de neurones de la i-ème couche, le i-ème élément de l est composé d'une matrice $W = (w_{jk})$ de nombres flottants à n lignes et m colonnes et d'un vecteur $H = (h_k)$ de nombres flottants de taille m. Si la valeur du neurone j de la couche $i-1$ (ou, pour $i = 0$, l'entrée j du réseau) est une entrée du neurone k de la couche i, w_{jk} est la valeur du poids correspondant ; w_{jk} vaut 0 sinon. Enfin, le nombre h_k est la valeur du seuil associé au neurone k de la couche i.

Ainsi, si l'exemple suivant représente deux couches consécutives (les nombres indiqués à côté des sommets étant les seuils associés, ceux à côté des arêtes les poids associés), la matrice associée à la seconde couche sera alors $\begin{pmatrix} 0 & 2 \\ -1 & 1 \\ 0 & 3 \end{pmatrix}$ et le vecteur $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$.



On introduit les types suivants :

```
type vecteur = float array
type matrice = float array array
type reseau_matrice = (matrice * vecteur) list
```

Proposer le code *OCaml* d'une fonction `eval: reseau_matrice -> vecteur -> vecteur` qui renvoie le résultat de l'évaluation d'un tel réseau de neurones sur un vecteur donné (ledit vecteur décrivant la valeur des entrées du réseau de neurones).

5 À propos des circuits booléens

Dans les circuits booléens tels que définis plus haut, on a considéré que $\mathcal{G}$, l'ensemble des portes élémentaires était $\mathcal{G} = \{\mathrm{AND}_2, \mathrm{OR}_2, \mathrm{NOT}_1, 0, 1\}$. On parlera de circuits booléen *généralisé* si l'on s'autorise $\mathcal{G} = \{\mathrm{AND}_n, \mathrm{OR}_n, \mathrm{NOT}_1, 0, 1 | n \in \mathbb{N}\}$: autrement dit, les portes *ET logique* et *OU logique* peuvent avoir plus que deux arguments. On appellera *porte* AND *généralisée* (respectivement : *porte* OR *généralisée*) une telle porte AND_n (respectivement : OR_n) pour un certain entier n.

Une formule en forme normale conjonctive (autrement dit sous la forme d'une conjonction de disjonctions de *littéraux*, un littéral étant une variable ou sa négation) peut s'écrire comme un circuit $NOT - OR - AND$: c'est-à-dire, comme un circuit généralisé avec :

— une unique sortie, étiquetée par une porte AND généralisée ;
— les entrées, ou les portes 0 et 1, qui ont leur(s) arc(s) sortant(s) qui vont soit vers une porte NOT_1, soit vers l'une des portes OR généralisées ;
— les portes NOT_1 qui ont leur(s) arc(s) sortant(s) qui va vers une porte OR généralisée ;
— les portes OR généralisées ont leur arc sortant qui va vers la porte AND généralisée.

Symétriquement pour une formule en forme normale disjonctive qui peut s'écrire comme un circuit $NOT - AND - OR$, défini en inversant le rôle des OR et AND dans la description précédente.

On appellera circuit 2-alternant un circuit qui est soit $NOT - OR - AND$, soit $NOT - AND - OR$. On souhaite prouver que tout circuit 2-alternant qui calcule la fonction PARITY a une taille au moins $2^{n-1} + 1$.

Question 20. Montrer que dans tout circuit $NOT - AND - OR$ qui calcule la fonction PARITY, pour chaque entrée $b = (b_1, \ldots, b_n) \in \mathbf{B}^n$ avec $\mathrm{PARITY}(b_1, \ldots, b_n) = 1$, il y au moins une porte AND généralisée, que l'on appellera A_b, dont la valeur de sortie sur l'entrée $(b_1, \ldots, b_n)$ est 1.

Question* 21. Montrer que dans la question précédente, on peut supposer de plus qu'il existe dans le circuit un chemin de chacune des n entrées vers cette porte A_b.

Tournez la page S.V.P.

Question 22. Montrer que tout circuit $NOT - AND - OR$ qui calcule la fonction PARITY a une taille au moins $2^{n-1} + 1$. En déduire que tout circuit 2-alternant qui calcule la fonction PARITY a une taille au moins $2^{n-1} + 1$.

6 À propos des réseaux de neurones à seuil à poids unitaires

Une fonction $f : \mathbf{B}^n \to \mathbf{B}^m$ est dite *symétrique* si sa valeur ne dépend pas de l'ordre de ses arguments.

Question 23. Soit $m \leq n$ des entiers. Proposer un neurone à seuil à poids unitaires qui renvoie 1 sur les entrées $x_1, \ldots, x_n \in \mathbf{B}$ si et seulement si au moins m de ces entrées ont la valeur 1. Proposer par ailleurs un neurone à seuil à poids unitaires qui renvoie 1 sur les entrées $x_1, \ldots, x_n \in \mathbf{B}$ si et seulement si au plus m de ces entrées ont la valeur 1.

Question 24. Montrer que toute fonction symétrique $f : \mathbf{B}^n \to \mathbf{B}$ peut être calculée par un réseau de neurones à seuil à poids unitaires de taille $\mathcal{O}(n)$ et profondeur 2.

Question 25. En déduire que l'on peut calculer la fonction PARITY avec une profondeur 2 et une taille $\mathcal{O}(n)$.

Rappelons que l'on a prouvé qu'on ne pouvait pas la calculer en profondeur 1. *C'est donc un exemple de fonction qui ne se calcule pas avec une couche, mais peut se calculer avec deux, et se calcule avec beaucoup moins de portes en utilisant des réseaux de neurones qu'avec des circuits booléens.*

Partie III. Complexité de l'apprentissage d'un circuit

Nous étudions dans cette partie la complexité de déterminer les poids d'un réseau de neurones dont l'architecture est fixée.

Appelons *architecture* un graphe comme dans la définition 1 (c'est-à-dire un circuit sans les étiquettes des sommets de V), avec n entrées et m sorties. Une *tâche* est alors un élément de $\mathbf{B}^n \times \mathbf{B}^m$.

Définition 2. *On dit que l'architecture est* compatible *avec la tâche $(x_1, \ldots, x_n, y_1, \ldots, y_m)$ s'il existe des valeurs des poids* dans $\{-1, 0, 1\}$ *et des valeurs* entières *des seuils pour les portes du graphe telles qu'on obtient la sortie $(y_1, \ldots, y_m)$ quand on évalue le réseau de neurones correspondant sur l'entrée $(x_1, \ldots, x_n)$.*

On s'intéresse au problème de décision **Apprentissage exact :** étant donnée une architecture et une liste finie de tâches, déterminer si l'architecture est compatible avec toutes les tâches de la liste au sens de la définition 2 (c'est-à-dire avec des poids unitaires et des seuils entiers).

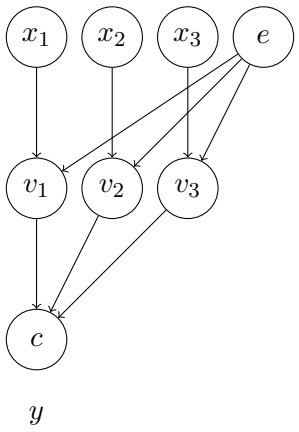
Question 26. Montrer que le problème est dans la classe NP.

On va chercher à prouver que le problème est NP-complet, en utilisant la NP-complétude de 3-SAT, que l'on admettra. On rappelle qu'il s'agit de déterminer la satisfiabilité d'une formule en forme normale conjonctive, avec 3 littéraux par clause (on rappelle qu'une *clause* est une disjonction de littéraux).

Question 27. On considère l'architecture à 4 entrées (à savoir x_1, x_2, x_3, e) et 1 sortie (à savoir y) représentée graphiquement ci-dessous.

On considère les 8 tâches que l'on obtient en faisant varier $x_1, x_2, x_3 \in \mathbf{B}$, et en ayant fixé $e = 0$, en prenant y qui vaut 1 sauf dans l'unique cas $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ où y vaut 0.

Prouver que l'architecture ci-dessous est compatible avec ces 8 tâches.



Question 28. On suppose qu'on a fixé les seuils et poids de l'architecture de la question précédente de manière à obtenir un réseau de neurones qui, sur les 8 tâches de cette dernière question, produit les sorties attendues. Notons $f_i(x)$ la valeur d'activation du nœud v_i sur l'entrée $(x, 0)$. Montrer que f_i est soit la fonction identité, soit la fonction $x \mapsto \mathrm{NOT}(x)$.

Question 29. On suppose toujours donné un réseau de neurones comme à la question précédente. Montrer que la valeur d'activation de c, notée y, s'exprime sous la forme $y = \mathrm{NOT}(x_1) \vee x_2 \vee \mathrm{NOT}(x_3)$, où l'entrée $x_i = f_i^{-1}(a_i)$ est soit a_i, soit $\mathrm{NOT}(a_i)$.

Nous avons donc montré que tout réseau construit par apprentissage exact à partir de l'architecture et des tâches données à la question 27 calcule la fonction $\mathrm{NOT}(x_1) \vee x_2 \vee \mathrm{NOT}(x_3)$.

Plaçons-nous maintenant dans le contexte où l'architecture et les tâches de la fonction 27 sont un sous-graphe et des sous-tâches d'une architecture $\mathcal{A}$ plus grande. Si l'on ajoute aux 8 tâches initiales une tâche de la forme $(x_1, x_2, x_3, y, e) = (\alpha_1, \alpha_2, \alpha_3, 1, 1)$ où $\alpha_1, \alpha_2, \alpha_3$ sont arbitraires, on déduit des questions précédentes que si l'architecture $\mathcal{A}$ est compatible avec cet ensemble de tâches, il existe des valeurs de x_1, x_2, x_3 telles que la clause $\mathrm{NOT}(x_1) \vee x_2 \vee \mathrm{NOT}(x_3)$ *est satisfaite.*

Question* 30. Déduire de ce qui précède que le problème de l'apprentissage exact des réseaux de neurones de profondeur 2 est NP-complet.

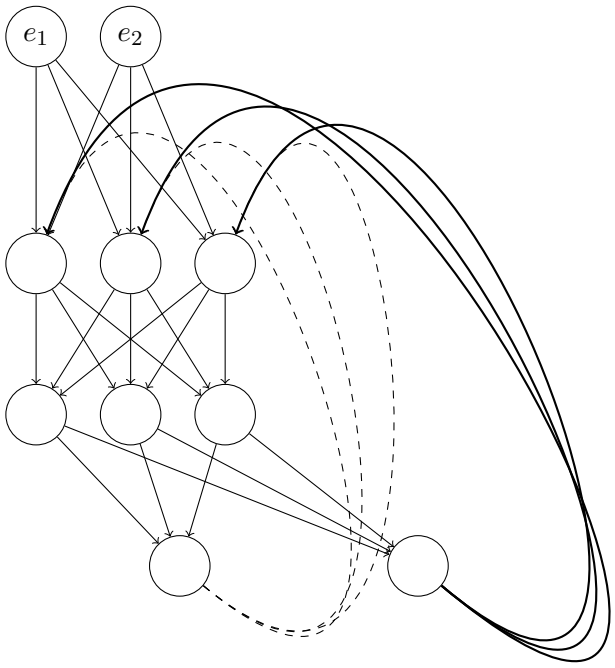
Tournez la page S.V.P.

Partie IV. Réseaux de neurones récurrents et automates finis

On considère maintenant des réseaux de neurones *récurrents*, c'est-à-dire où l'organisation en couche est cyclique. On obtient un tel réseau de la façon suivante : on part d'un réseau de neurones R bien formé avec $p+n$ entrées, et n sorties, pour deux entiers n et p : ses entrées sont numérotées de 1 à $n+p$ et ses sorties de 1 à n. On identifie les n sorties avec les n dernières entrées : c'est-à-dire que chaque arc sortant de l'entrée numéro $p+i$ vers un certain sommet s devient un arc sortant de la sortie numéro i vers le sommet s. Il reste p entrées que l'on considère comme les p entrées du réseau récurrent.

Dans un réseau récurrent, les valeurs d'activations des neurones évoluent au cours du temps : à chaque pas de temps, le résultat des n sorties au temps t sont réinjectées comme entrées à l'étape $t+1$.

Par exemple, en partant du réseau R représenté en haut de la page 23, en considérant $p=2$ et $n=2$, on obtient l'architecture suivante (les différents types des arêtes, ordinaire, épais, et pointillés, sont là pour améliorer la lisibilité du graphe et n'ont pas de sémantique particulière).



Si le graphe de R était le graphe acyclique $(V \cup X, E)$ (voir la définition 1), alors le graphe obtenu s'écrit $(V \cup X', E')$ avec X' constitué de p sommets, qui correspondent aux entrées restantes. On peut supposer que V s'écrit $V = \{1, 2, \ldots, m\}$ où m est le nombre de sommets de V. À un instant t on décrit les valeurs d'activation de chacun des neurones de V par le vecteur $\mathbf{z}(t) = (z_1(t), z_2(t), \ldots, z_m(t))$. Initialement, $z_i(0) = 0$ pour tout $1 \leq i \leq m$.

À chaque instant $1 \leq t$, on note $x_1(t), x_2(t), \ldots, x_p(t)$ la valeur des entrées au temps t. La valeur de $z_j(t+1)$ est alors obtenue comme

$$z_j(t+1) = \mathcal{A}\left(\sum_{1\leq i\leq p:(i,j)\in E'} w_{i,j}x_i(t) + \sum_{i'\in V:(i',j)\in E'} w_{i',j}z_{i'}(t) - h_j\right),$$

où $w_{i,j}$ désigne le poids correspondant à l'arc de l'entrée x_i vers le sommet j, et $w_{i',j}$ désigne le poids de l'arc du sommet i' de V vers le sommet j, et h_j le seuil du sommet j. Cela correspond à propager les valeurs selon la dynamique de chacun des neurones de façon synchrone (c'est-à-dire tous en même temps).

On va chercher à caractériser ce que l'on peut calculer avec de tels réseaux.

7 Réseaux de neurones à seuil et reconnaissance immédiate

On rappelle qu'un *automate fini* (déterministe) est la donnée de $(Q, \Sigma, q_0, A, \delta)$: Q est l'ensemble fini de ses états, Σ est son alphabet fini d'entrée, $q_0 \in Q$ est son état initial, $A \subseteq Q$ est son ensemble des états acceptants, et $\delta : Q \times \Sigma \to Q$ est sa fonction de transition. On suppose que $\Sigma = \mathbf{B} = \{0, 1\}$ est l'alphabet de tous les mots et langages mentionnés dans la suite.

À l'automate fini M et un mot $\omega = a_1 \ldots a_\ell$, avec chaque $a_i \in \Sigma$, on associe la suite d'états définie par $q(0) = q_0$, et $q(t+1) = \delta(q(t), a_t)$. Le mot ω est accepté si $q(\ell) \in A$. Le langage $L(\mathcal{M}) \subseteq \Sigma^*$ accepté par $\mathcal{M}$ est l'ensemble des mots acceptés. Un langage est *régulier* s'il est accepté par un automate fini.

On admettra que l'on pourrait considérer qu'il n'y a qu'un seul état acceptant (autrement dit formellement supposer que A est un singleton), et qu'il n'est pas possible de faire une transition vers l'état initial q_0 (autrement dit, $\delta(r, s) \neq q_0$ pour tout $r \in Q, s \in \Sigma$) : cela ne change rien à la classe des langages acceptés. On note q le nombre d'éléments de Q.

On peut considérer que Q, l'ensemble des états, est donné comme $\{\mathbf{e}_1, \ldots, \mathbf{e}_q\} \subseteq \mathbf{B}^q$, où le vecteur $\mathbf{e}_i$ est le vecteur de $\mathbf{B}^q$ dont toutes les coordonnées sont nulles, sauf la i-ème qui vaut 1.

Question 31. Montrer que l'on peut construire un réseau de neurones [2] à seuil à poids unitaires, avec $q+1$ entrées et q sorties, qui calcule cette fonction.

Introduisons un encodage alternatif qui rend possible la construction d'un réseau de neurone de profondeur 1 : on utilise $2q$ variables $x_{i,e}$ pour $1 \leq i \leq q$, et $e \in \mathbf{B}$. Chaque variable $x_{i,e}$ prend ses valeurs dans $\mathbf{B}$, et vaut 1 si et seulement si l'état de l'automate est q_i et le dernier symbole lu est le symbole e. À tout instant $t \geq 1$, exactement une de ces variables vaut 1, et toutes les autres valent 0. La fonction $\delta : Q \times \mathbf{B} \to Q$ peut alors se voir comme une fonction [3] $\mathbf{B}^{2q+1}$ dans $\mathbf{B}^{2q}$ qui met à jour ces variables.

2. Non-récurrent.
3. fonction partielle, au sens où elle n'est potentiellement définie que sur un sous-ensemble de $\mathbf{B}^{2q+1}$.

Tournez la page S.V.P.

Question 32. Montrer que l'on peut construire un réseau de neurones [4] à seuil, de profondeur 1, à poids unitaires, avec $2q+1$ entrées et $2q$ sorties, qui calcule cette fonction.

On veut considérer un réseau de neurones récurrent comme *reconnaissant un langage* : on considère qu'un des neurones est *le neurone de décision*. On suppose $p = 1$ (une unique entrée). On note $\omega = x_1(1) \cdot x_1(2) \cdot \ldots \cdot x_1(\ell)$ le mot de taille ℓ sur l'alphabet Σ obtenu en concaténant la valeur de l'entrée $x_1(t)$ aux temps $t = 1, \ldots, \ell$. On dit que le mot ω est *accepté* si au temps ℓ, le neurone de décision a sa valeur d'activation à 1, *rejeté* sinon.

Question 33. Prouver que si un langage est régulier, alors il correspond à l'ensemble des mots acceptés par un réseau de neurones récurrent à seuil.

Question 34. Prouver la réciproque : tout langage accepté par un réseau de neurones récurrent à seuil est régulier.

8 Réseaux de neurones à seuil et reconnaissance hors ligne

Dans un automate fini, la décision d'accepter (ou de rejeter) est prise immédiatement à la fin de la lecture du mot. On considère un modèle d'automate fini *hors-ligne*, c'est-à-dire que le calcul dans l'automate peut continuer après la fin de la lecture du mot reçu en entrée ; la décision d'acceptation ou de rejet intervient alors à la fin du calcul.

Formellement, un *automate fini hors-ligne* est la donnée de $(Q, \Sigma, q_0, A, \delta)$ définis exactement comme pour un automate fini, si ce n'est que sa fonction de transition f n'envoie pas $Q \times \Sigma$ sur Q, mais $Q \times (\Sigma \bigcup \{\$\})$ sur Q, où \$ est un symbole spécial ($\$ \notin \Sigma$) qui encode le fait que le mot en entrée a été complètement lu par la machine. Ici encore, on suppose toujours que $\Sigma = \mathbf{B}$.

À l'automate fini hors-ligne M et un mot $\omega = a_1 \ldots a_\ell$, avec chaque $a_i \in \Sigma$, on associe la suite d'états définie par $q(0) = q_0$, et $q(t+1) = \delta(q(t), a_t)$ pour $t = 1, 2, \ldots \ell$, puis $q(t+1) = \delta(q(t), \$)$ pour $t \geq \ell$. Le mot ω est accepté s'il existe $t \geq \ell$ tel que $q(t) \in A$.

Question 35. Montrer qu'un langage est régulier si et seulement s'il est accepté par un automate fini hors-ligne.

On veut considérer un réseau de neurones récurrent comme *reconnaissant un langage hors ligne* : on considère $p = 2$: le réseau récurrent possède deux entrées, l'une appelée D, disons x_1, qui sert à entrer les données, et l'autre V, disons x_2, qui sert à indiquer quand l'entrée est valide (ou si l'on préfère, de façon duale, à dire quand on a terminé de donner l'entrée). Jusqu'à un certain temps ℓ, V vaut 1, puis ensuite V est maintenu à 0. La valeur de l'entrée D au temps $t = 1, 2, \ldots, \ell$, peut se voir comme un mot ω de longueur ℓ sur l'alphabet $\Sigma = \mathbf{B}$.

On considère qu'un des neurones est le *neurone de décision* G, et qu'un autre neurone est le *neurone de validation* H. On dit que le réseau *calcule* tant que H est à 0.

4. Non-récurrent.

Soit $\omega \in \Sigma^*$. On définit le temps d'arrêt T_ω comme le plus petit entier (éventuellement infini) t tel que $H(t) = 1$, sur l'entrée ω. Le mot ω est accepté si T_ω est fini et $G(T_\omega)$ vaut 1.

Question 36. Prouver que si un langage est régulier alors il correspond à l'ensemble des mots acceptés par un réseau de neurones récurrent à seuil hors ligne.

La réciproque est vraie : tout langage accepté par un réseau de neurones récurrent à seuil hors ligne est régulier. C'est essentiellement le même raisonnement que dans la question 34.

9 Réseaux de neurones linéaires saturés à coefficients entiers

On s'intéresse dans cette question, et dans les parties suivantes à des circuits de neurones (réseaux de neurones) pour lesquels la fonction d'activation $\mathcal{A}$ n'est pas nécessairement la fonction de Heaviside. Étant donnée une certaine fonction $\mathcal{A}$ sur K, on considère les circuits que l'on obtient en prenant pour $\mathcal{G}$ l'ensemble des neurones (un neurone à n entrées ayant l'arité n) construits à partir de cette fonction d'activation $\mathcal{A}$.

En particulier, on appelle *sigmoïde idéale* la fonction continue $\sigma(x)$ qui vaut x pour $x \in [0, 1]$, 0 pour $x \leq 0$, et 1 pour $x \geq 1$. On parle de neurone *linéaire saturé* pour un neurone construit en prenant cette fonction σ comme fonction $\mathcal{A}$ sur $K = \mathbb{R}$. On parle de réseaux de neurones *linéaires saturés* lorsque tous les neurones du réseau le sont.

Question 37. Soit L un langage sur l'alphabet $\mathbf{B} = \{0, 1\}$. Prouver que les assertions suivantes sont équivalentes :

1. Le langage L est régulier.
2. Le langage L est accepté par un réseau de neurones linéaire saturé dont les poids et seuils sont entiers.
3. Le langage L est accepté hors ligne par un réseau de neurones linéaire saturé dont les poids et seuils sont entiers.

Partie V. Réseaux de neurones récurrents ReLU

Dans cette section, on considère cette fois des réseaux de neurones où la fonction $\mathcal{A}$ est la fonction *ReLU* (pour *Rectified Linear Unit* en anglais, soit *Unité Linéaire Rectifié* en français), c'est-à-dire la fonction continue $\mathcal{R}(x) = \max(x, 0)$ qui vaut x pour $x \geq 0$, et 0 pour $x \leq 0$. On parle de neurone ReLU pour un neurone construit en prenant cette fonction $\mathcal{R}$ comme fonction $\mathcal{A}$ sur $K = \mathbb{R}$, et on parle de réseau de neurones ReLU lorsque tous les neurones du réseau le sont.

Une *machine à compteurs* possède un compteur d'instruction R et un nombre fini k de compteurs $r_1, r_2, \cdots, r_k$, qui prennent leurs valeurs dans l'ensemble $\mathbb{N}$ des entiers naturels. L'état de la machine à un instant donné est donné par la valeur du $k+1$-uplet d'entiers $(R, r_1, \ldots, r_k) \in \mathbb{N}^k$.

Tournez la page S.V.P.

Un programme d'une telle machine est constitué d'une liste finie $I_1, I_2, \ldots, I_q$ d'instructions. Chaque instruction est de l'un des trois types suivants :

— $\mathrm{Inc}(c, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(j, r_1, \ldots, r_{c-1}, r_c + 1, r_{c+1}, \ldots, r_k)$. Autrement dit, on incrémente le compteur c et on va à l'instruction j.
— $\mathrm{Decr}(c, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(j, r_1, \ldots, r_{c-1}, \max(0, r_c - 1), r_{c+1}, \ldots, r_k)$. Autrement dit, on décrémente le compteur c s'il était strictement positif, on le laisse inchangé sinon, et on va à l'instruction j.
— $\mathrm{IsZero}(c, i, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq i \leq q$, $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(i, r_1, \ldots, r_k)$ si $r_c = 0$ et $(j, r_1, \ldots, r_k)$ sinon. Autrement dit, on teste si le compteur c vaut 0, et on va à l'instruction i si c'est le cas, à l'instruction j sinon.

On fixe une valeur initiale pour les compteurs $r_1, \ldots, r_k$. Le compteur d'instruction R vaut initialement 1. On convient que lorsque $R = 0$, la machine s'arrête. À chaque instant t, tant que $R \neq 0$, on regarde la valeur du compteur d'instruction R. On exécute alors l'instruction I_R correspondante, qui met à jour R, et les valeurs des compteurs $r_1, \ldots, r_k$ selon les règles plus haut pour cette instruction I_R.

Question 38. Décrire un programme d'une machine à deux compteurs qui multiplie par 2, c'est-à-dire qui, sur l'entrée $(1, n, 0)$ arrive, après un certain nombre d'étapes, dans l'état $(0, 2n, 0)$.

Question 39. Montrer que l'on peut construire un neurone ReLU (c'est-à-dire dont la fonction d'activation est la fonction ReLU) avec une entrée et une sortie, et qui calcule la fonction qui à $r \in \mathbb{R}$ associe $r + 1$. Même question pour la fonction qui à $r \in \mathbb{R}$ associe $\max(0, r - 1)$. Même question pour la fonction qui à $r \in \mathbb{R}$ associe 1 si $r = 0$ et 0 sinon.

Question 40. On fixe trois entiers c, u, v et l. Décrire un programme d'une machine à compteurs (avec k suffisamment grand) tel que, lorsque la machine démarre dans l'état initial $(1, r_1, \ldots, r_k)$, elle se retrouve après un certain temps dans l'état $(l, r_1, \ldots, r_{c-1}, r_u + r_v, r_{c+1}, \ldots, r_k)$.

En utilisant un tel programme, on peut par conséquent considérer que le jeu d'instruction n'est pas limité aux trois types d'instructions plus haut, mais peut également être une instruction du type $\mathrm{Add}(c, u, v, l)$*, pour certains* $1 \leq c \leq k$*,* $1 \leq u \leq k$*,* $1 \leq v \leq k$*,* $0 \leq l \leq q$ *: si l'état de la machine est* $(R, r_1, \ldots, r_k)$*, il devient* $(l, r_1, \ldots, r_{c-1}, r_u + r_v, r_{c+1}, \ldots, r_k)$*.*

De même on peut autoriser une instruction du type $\mathrm{Sub}(c, u, v, l)$*, pour certains* $1 \leq c \leq k$*,* $1 \leq u \leq k$*,* $1 \leq v \leq k$*,* $0 \leq l \leq q$ *: si l'état de la machine est* $(R, r_1, \ldots, r_k)$*, il devient* $(l, r_1, \ldots, r_{c-1}, \max(0, r_u - r_v), r_{c+1}, \ldots, r_k)$*.*

Question 41. On considère un neurone ReLU unitaire : pour certains poids $w_1, w_2, \ldots, w_n$ dans $\{-1, 0, 1\}$, et un seuil h entier, il calcule la fonction qui aux entiers naturels $x_1, \ldots, x_n$ associe $\mathcal{R}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - h)$.

Décrire un programme d'une machine à compteurs qui calcule cette fonction.

On peut par conséquent se convaincre que pour tout réseau de neurones récurrent ReLU à poids unitaires, on peut construire une machine à compteur qui le simule.

Question 42. Prouver la réciproque : montrer que pour tout programme de machine à compteurs, on peut construire un réseau ReLU à poids unitaires qui le simule. On précisera en particulier comment est codé le compteur d'instruction, et sa mise à jour. On s'autorisera à ce que la simulation ne soit pas *en temps réel* : t instructions sont simulées par $t' > t$ étapes (c'est-à-dire $t' > t$ propagations des valeurs selon la dynamique) du réseau : on précisera la relation entre t et t' dans la simulation proposée.

Autrement dit, sur les entrées entières, les réseaux de neurones récurrents ReLU à poids unitaires et seuils entiers ont la même puissance que les machines à compteurs. On admettra que cela correspond également à celle des machines de Turing.

Partie VI. Réseaux de neurones récurrents linéaires saturés

Dans cette section, on considère à nouveau des réseaux de neurones où la fonction $\mathcal{A}$ est la fonction *sigmoïde idéale*, c'est-à-dire la fonction continue $\sigma(x)$ qui vaut x pour $x \in [0,1]$, 0 pour $x \leq 0$, et 1 pour $x \geq 1$. On rappelle que l'on parle de neurones ou de réseaux de neurones *linéaires saturés* dans ce cas.

Question 43. Montrer que toute fonction calculée par un réseau de neurones linéaire saturé se calcule par un réseau de neurones ReLU (de taille et profondeur plus grande).

La réciproque est fausse car la fonction $\mathcal{R}$ peut prendre des valeurs en dehors de $[0,1]$ que l'on ne peut pas générer avec la fonction σ. Les résultats précédents donnent un moyen de simuler une machine de Turing (un calcul arbitraire) par un réseau ReLU, mais on utilise des neurones dont la valeur d'activation peut devenir arbitrairement grande. Par ailleurs, la simulation de t étapes d'une machine de Turing se fait en pratique avec un nombre non polynomial en t de mise à jour du réseau de neurones. On va prouver que l'on peut rester sur un domaine borné, en l'occurrence $[0,1]$, en utilisant les réseaux de neurones linéaires saturés, et par ailleurs de façon beaucoup plus efficace en temps.

On appelle *pile* une variable r qui prend ses valeurs dans l'ensemble $\mathbf{B}^*$ des mots sur l'alphabet $\mathbf{B}$. Autrement dit, à un instant donné, r s'écrit $r = w_1 w_2 \dots w_n$ avec chacun des $w_i \in \mathbf{B}$. On note $push_0 : \mathbf{B}^* \to \mathbf{B}^*$ la fonction qui envoie r sur le mot $0r$, soit le mot qui s'écrit $0w_1 w_2 \dots w_n$. On note $push_1 : \mathbf{B}^* \to \mathbf{B}^*$ la fonction qui envoie r sur le mot $1r$, soit le mot qui s'écrit $1w_1 w_2 \dots w_n$. On note $pop : \mathbf{B}^* \to \mathbf{B}^*$ la fonction qui envoie r sur le mot obtenu en enlevant sa première lettre soit $w_2 \dots w_n$, ou le mot vide si r était le mot vide. On note $top : \mathbf{B}^* \to \mathbf{B}$ la fonction qui envoie r sur sa première lettre w_1 (ou sur 0 si r était le mot vide).

Question 44. On code chaque mot $r = w_1 \dots w_n$ sur l'alphabet $\Sigma = \mathbf{B}$ par le rationnel de $[0,1]$ défini par $\gamma(r) = \sum_{i=1}^{n} \frac{2w_i+1}{4^i}$.

Montrer qu'on peut construire un neurone linéaire saturé qui simule l'effet de *top* sur une pile : si on lui donne en entrée $\gamma(r)$, alors sa sortie (valeur d'activation) correspond à $top(r)$. Montrer qu'on peut construire un neurone linéaire saturé qui simule l'effet de $push_0$ sur une

Tournez la page S.V.P.

pile : si on lui donne en entrée $\gamma(r)$, alors sa sortie correspond à $\gamma(push_0(r))$. Même question pour $push_1$ et pop.

Une *machine à piles* possède un compteur d'instruction R et un nombre fini k de piles $r_1, r_2, \cdots, r_k$. L'état de la machine à un instant donné est donné par la valeur du $k+1$-uplet d'entiers $(R, r_1, \ldots, r_k) \in \mathbb{N} \times (\mathbf{B}^*)^k$.

Un programme d'une telle machine est constitué d'une liste finie $I_1, I_2, \ldots, I_q$ d'instructions. Chaque instruction est de l'un des quatre types suivants :

— $\text{Push}_0(c, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(j, r_1, \ldots, r_{c-1}, push_0(r_c), r_{c+1}, \ldots, r_k)$.
— $\text{Push}_1(c, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(j, r_1, \ldots, r_{c-1}, push_1(r_c), r_{c+1}, \ldots, r_k)$.
— $\text{Pop}(c, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(j, r_1, \ldots, r_{c-1}, pop(r_c), r_{c+1}, \ldots, r_k)$.
— $\text{Top}(c, i, j)$, pour un certain $1 \leq c \leq k$ et $0 \leq i \leq q$, $0 \leq j \leq q$: si l'état de la machine est $(R, r_1, \ldots, r_k)$, il devient $(i, r_1, \ldots, r_k)$ si $top(r_c) = 0$ et $(j, r_1, \ldots, r_k)$ si $top(r_c) = 1$.

On fixe une valeur initiale pour les piles $r_1, \ldots, r_k$. Le compteur d'instruction R vaut initialement 1. On convient que lorsque $R = 0$, la machine s'arrête. À chaque instant t, tant que $R \neq 0$, on regarde la valeur du compteur d'instruction R. On exécute alors l'instruction I_R correspondante, qui met à jour R, et les valeurs des piles $r_1, \ldots, r_k$ selon les règles plus haut pour cette instruction I_R.

Question 45. Montrer que toute machine de Turing peut être simulée par une machine à 2 piles.

Question 46. Prouver que pour toute machine de Turing, on peut construire un réseau de neurones linéaires saturés dont tous les coefficients sont rationnels qui la simule. On s'autorisera à ce que la simulation ne soit pas *en temps réel*[5].

La réciproque est vraie : tout réseau de neurones linéaires saturés dont tous les coefficients sont rationnels peut être simulé par une machine de Turing. Cela découle de la thèse de Church-Turing, ou peut se prouver en construisant explicitement un programme de machine de Turing adéquat.

Les réseaux de neurones linéaires saturés dont les coefficients sont des rationnels sont donc essentiellement des machines de Turing, en terme de puissance de calcul.

Question 47. Formaliser le problème de l'apprentissage exact d'un réseau de neurones linéaires saturés. Prouver que le problème est indécidable.

On remarquera que l'on cherche souvent dans le contexte des réseaux de neurones, et de l'apprentissage profond *à apprendre un réseau de façon approchée, alors que ces résultats concernent*

5. Voir l'énoncé de la question 42.

la question de l'apprentissage exact *(dans le sens où on cherche un réseau qui correspond* exactement *aux exemples).*

Question 48. Le problème de l'apprentissage exact reste-t-il indécidable si on fixe une architecture ? Pour un réseau de neurones dont on fixe l'architecture ainsi que tous les poids et seuils, sauf un ? Discuter ce que l'on obtient.

* *
*