* Deux model qui se bagarre : Time based sharing model TBRS
* Modèle de la mémoire de travail à la frontière entre psycho et math
  + Intégré des math = prédiction

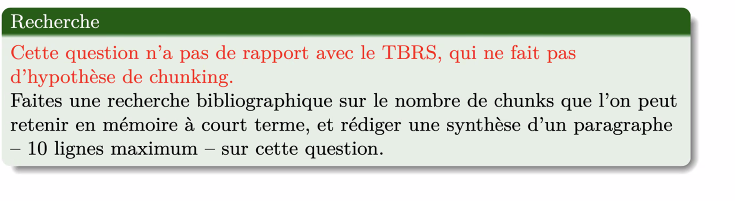
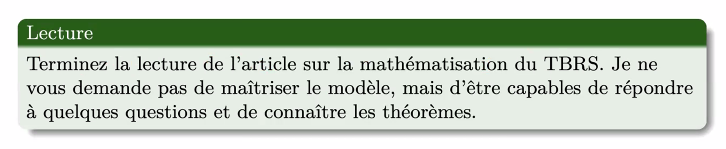
Principes du TBRS :

* Comment les gens font pour rappeler les lettres en mémoire de travail avec une tâche intermittente secondaire ?
  + 3 évents : présentation de lettre, tache concurrente/secondaire, temps libre
  + Effet : lettrer → temps → tache : rappel ++ // lettre → tache → temps : rappel –
  + Ici on prend en compte le temps suffit à expliquer l’effacement en mémoire
* Principe de base du TBRS :
  + A chaque instant l’attention est focalisée soit sur **une** lettre à rappeler (un item), soit sur la tâche secondaire
  + Lorsque l’attention est focalisée sur un item, l’**activation** de l’item augmente. Sinon, elle décroit
  + Cette loi de croissance ou de décroissance ne dépend pas des autres items ou du temps en soi, mais seulement de l’activation actuelle
    - Temps : même augmentation au début et à la fin de l’expérience (pas d’effet de fatigue ou quoi)
    - De l’item : same pour chaque item
    - Mais la façon dont ça monte ou ça descend peut-être choisi, une exp, un logarithmique
  + La probabilité de rappel correct d’un item à un moment donnée est une fonction croissante de l’activation
    - Plus l’item est activé, plus y’a de chance de le rappeler
  + Le **cognitive load** est la proportion du temps passée à ne pas se focaliser sur un item. L’activation dépend du cognitive load et du temps total, mais pas de la répartition des moment passés à réactiver
    - Exemple : lettre → 10s → tache → rappel <=> lettre → 3s → tache → 7s → rappel
* L’article :
  + Complex span task : présentation d’item + tache secondaire à effectuer
  + 3 taches concurrentes utilisé : lire des lettres, vérifier des calculs mathématiques, update un chiffre en mémoire, répétition de syllabe.
  + Cognitive load : la proportion du temps passée à ne **pas** se focaliser sur un item.
  + Critiques TBRS\* : Connexionniste
    - S’éloigne du TBRS original en intégrant d’autre modèle
    - Intégration non nécessaire de certaine chose (eg. Serial position coding)
      * Choix de l’implémentation de comment coder les lettres dans le modèle
      * Rajout d’hypothèse par rapport au TBRS
    - Modèle connexionniste = boite noire, comme en deep learning, on a juste des poids de neurone
    - Beaucoup de paramètre en plus par rapport au TBRS originel. Moins de paramètre est préférable. Sinon on modélise tous les cas possibles etc.
* Démontrer que si les HP du TBRS sont vérifiées (strictement), alors il n’y a jamais d’oubli possible
  + HP du cognitive load
  + Exemple :
    - A → tache
    - → baisse de l’activation : jusqu’avant l’oublie
    - → temps libre : jusqu’à qu’on retourne au niveau de base
    - → boucle : ainsi de suite
    - On n’oublie pas
  + Est-ce qu’en 30 s
  + Pas compris pourquoi
  + Mais apparemment c’était vraiment comme la demi-vie

Probabilité, chances et log-odds

* Odds = les chances,
* Valeur possible :
* Multiplier la chance par deux ne revient pas à multiplier la proba par deux

Fonction de tache, de focus et d’activation

* Lecture de la part Overview de l’article
  + Comment est définie l’activation ? par la log-odds
  + Q2
  + Q3 : Dépend de la durée du focus, définis l’alternance entre les deux ou n lettre. Combien de temps on reste focus sur un item avant de passer au suivant, fixe ou variable
  + Q4 : pas nécessairement dans l’autre
  + Q5 : dépends des taux de rafraichissement et ; à quelle vitesse sa croit/décroit
* Autre stratégie possible
* ……
* Faire un petit paragraphe sur le nombre de Chuck en mémoire DEMANDER A JOZE
* Lire l’article et avoir l’idée de comment ça fonctionne, comment ça a été mathématisé

Devoir :

Une chose est sûre, c’est que la mémoire à court terme à une capacité limité. On mesure généralement cette tâche par une tache de mémorisation-restitution. George Miller a suggéré en 1956 que la mémoire de travail avait une capacité de 7+-2, et cela s’accordé bien avec les découverte et recherche de l’époque. De nos jour, ce chiffre est à nuancer. En effet il correspond bien à la capacité des étudiants (participant majoritaire des études de psychologie) à retenir des suite de chiffre. Mais le matériel et la population testé exercent une influence sur le résultats. En effet des mots phonologiquement proche ou des mots long à prononcer sont plus difficile à restituer.   
Un chiffre plus précis aujourd’hui est de 4 chunks ou pièce d’information. Mais certaine théorie argumente contre une mesure pas un nombre d’éléments.

Article :

New Cours du 22/02/2022 :

Machine de Turing : on a fait plein de truc

La complexité algorithmique d’une chaine de caractère c’est la longueur du plus court programme qui permet de la mettre sur une machine de Turing universelle.

Démontrer que : Si alors

Exo 2 : où ne dépend que de T

Exo 3 :  : k représente le programme qui « émule » dans l’autre machine (exemple mac vs windows)

Probabilité algorithmique : on prend un programme au hasard et on regarde les chances qu’il nous donne une suite donnée fixé choisis

Coding théorème : fait le lien entre complexité et probabilité algorithmique

A un grand O prêt C(s) = -log2(m(s)) + O(1).

Problème de la machine de Turing :

* Ça dépend de la machine universelle choisis
* Ce n’est pas calculable, pas d’algo pour le trouver

Il existe un packet pour estimer ça : ACSS. Il a expliqué mais bon

* Je crois on fait tourner plein plein de machine de turing aléatoire et on regarde en moyenne ce que ca donne

L’entropie (compression) ne dépend uniquement du nombre de 0 et de 1 de la suite

* 011001 -> entropie max
* // 000000 -> Entropie minimale
* 010101 -> entropie same que le premier alors que d’un point de vu complexité d’algo c’est instinctivement plus simple

New Cours :

* Dans le modèle TBRS, quel élément détermine majoritairement le span ?
  + Les interférences avec les autres items
  + La vitesse de présentation des items
  + **Le cognitive load**
  + Le temps libre pour la répétition des items
* Le TBRS\* est une modélisation mathématique du modèle TBRS. Sur quoi repose cette modélisation ?
  + La statistique bayésienne
  + L'analyse complexe
  + La complexité algorithmique
  + **Les réseaux de neurones formels**
* Image :
  + 1ère partie : c’est la tache
  + 2ème  : focus fonction
  + 3ème : activation fonction
  + Chart

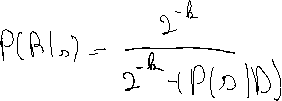
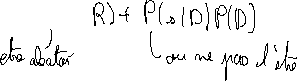
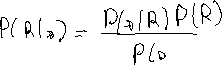
    Description automatically generated
* Selon le TBRS, quelle situation permet un meilleur rappel de 5 lettres présentées au départ
  + Les deux sont identique (selon le TBRS
  + IRL quand on a du temps après c’est mieux (consolidation) mais on n’est pas sûr à 100%
* Quelle durée approximative est souvent citées concernant le temps passé à rafraîchir un item dans les tâches de mémoire de travail ?
  + Environ 1/10 de seconde, mais ça varie beaucoup d’un auteur à un autre, mais c’est ce qui reviens le plus souvent
  + Implémenté dans le TBRD 2 comme paramètre
* Quelle différence peut-on faire entre "chunking" et "grouping" ?
  + Graphical user interface, text, application, email

    Description automatically generated
* Pour expliquer les variations de span lié au chunking, plusieurs hypothèses ont été avancées. Cocher celle qui n'est pas évoquée dans l'article de Norris
  + On remplace en mémoire le chunk par un pointeur vers la MLT
  + **En réalité, les chunks ne sont pas codés du tout en mémoire à court terme, puisqu'ils sont dans le mémoire à long terme.**
  + On recode en mémoire le chunk en utilisant des indices permettant de retrouver une trace dégradée
  + On seconde l'ensemble du mémorandum en utilisant un code utilisant la fréquence ou la familiarité des mots
* Cette machine est-elle une machine de Turing universelle. Que pouvez-vous en dire avec les informations que nous avons vu en séance ?
  + Diagram

    Description automatically generated
* Cocher les bonnes réponses
  + **Si une machine de Turing peut faire un calcul, alors il existe une infinité de machines de Turing pouvant le faire**
  + Il existe une machine de Turing qui peut renvoyer la complexité algorithmique d'une suite qu'on lui fournit en entrée
  + **Il existe une machine de Turing qui calcule les 1000 premières décimales de pi**
  + **Il existe une machine de Turing à 15 état qui calcul les 1000 premières décimales de e**
  + La probabilité algorithmique augmente avec la complexité algorithmique
    - Elle diminue
* La complexité algorithmique d'une suite (cochez les réponses exactes)
  + **Tend vers l'infini si la taille de la suite tend vers l'infini** (vu en cours démonstration)
  + **Dépend de la machine de Turing universelle choisie** (dépend d’une constante)
  + **A tendance à augmenter avec la longueur de la suite**
  + Est le temps qu'il faut pour calculer la suite
    - Non c’est la profondeur logique de Bennet, c’est le temps qu’il faut pour calculer la suite avec le programme **le plus court**
    - On peut faire un programme qui estime pi avec un séries VS un autre qui print simplement
  + Augmente moins vite que le carré de la longueur de la suite
    - Car c’est à une constante près ?
* Le "coding theorem" indique que si m désigne la probabilité algorithmique et C la complexité d'une suite, alors C(s) = -log2(m(s)) + O(1). Cela indique
  + Que la complexité augmente avec la probabilité algorithmique
  + **Que l'on peut estimer la complexité algorithmique à partir de la probabilité algorithmique**
  + Que la complexité est inférieure au logarithme de la probabilité
* Quel est un bon exemple de suite qui a une faible complexité, mais une grande profondeur logique de Bennett ? (on cherche un calcul court mais très long)
  + La suite des nombres impairs
  + Une suite aléatoire
  + La suite de Fibonacci
  + **La suite des nombres premiers**
* Quelle est la suite binaire la plus difficile à mémoriser ?
  + **XXOXOOOXOO**
  + OOXXOOXXOOXX
  + XOOXXX

Cours complexité :

* Test sur l’intuition de ce qui est complexe ou pas
  + Nous avons une intuition de la complexité (ou du hasard) partiellement conforme à ce que mesure la complexité algorithmique (de Kolmogorov)
    - Aléatoire = complexe
  + Il faut inhiber ce qu’il vient automatiquement pour produire des suite aléatoire (de chiffre ou lettre, ou pile ou face)
* Le vrai hasard
  + On va définir une sorte de mesure continue (pas discrète : aléatoire ou pas aléatoire) pour mesurer l’aléatoire
  + Exemple :
    - « PFPFPFPFPF » est une suite probable lorsque qu’on essaye de **pas** être aléatoire
    - Si on demande de dire une suite Pile ou Face à quelqu’un et qu’il nous sort ça on n’aura pas l’impression qu’il l’a dite aléatoirement
    - Alors que pourtant statistiquement elle a la même proba se sortir que n’importe qu’elle autre suite
  + Exercice
    - Soit s une suite binaire de longueur k fixé. La suite peut être produite pas un vrai processus aléatoire (évènement R) ou par un processus déterministe inconnu (autrement dit un programme événement D)
      * Calculez P(s|R)
        + 0 car on a une infinité de suite possible comme une densité continue donc P(X=S) = 0 => NON suite de taille fixé
        + 1/ \binom{2}{k} non plus, ça me saoule les proba discrète ptdr
        + C’était
      * De quelle notion P(s|D) peut être rapprochée ?
        + De la ~~complexité~~ algorithmique
        + De la probabilité algorithmique
        + Car si on sait que s a été produit avec un algo alors c’est en lien avec la proba de retrouver la suite en testant plusieurs algos aléatoirement
      * Exprimez le vrai hasard de s défini comme P(R|s)



* Graphique Vrais hasard en x et hasard subjectif
  + ACSS est cohérent avec le hasard subjectif
* **Notre perception du hasard peut se comprendre de manière rationnelle**
* Comment peut-on apprendre à déterminer si une suite est aléatoire sachant que la complexité algorithmique est non calculable ?
  + Pas d’algo simple pour obtenir la complexité (ACSS fait tourner 50000 machines de Turing)
  + Comment expliquer qu’on a un sens de la probabilité algorithmique ?
  + On apprend en regardant le monde (suivant des loi précise, ou des algos) que c’est sensé refléter la probabilité algorithmique
  + Donc c’est sensé se développer avec le temps, chez les enfants
* Mémorisation et complexité
  + Tache de jeux de Simon : Lorsque la complexité augmente la probabilité de rappel se réduit
  + On compresserait donc automatiquement d’une manière optimale ?
    - Non car est-ce que ce n’est pas juste un effet de la longueur !
  + Regardons à longueur égale alors :
    - On va quand même dans le sens que la complexité joue un rôle et pas uniquement la longueur
  + -> On doit compresser en MCT directement avec une méthode assez bonne pour s’approcher de la méthode optimale
  + -> La complexité algo d’une suite est meilleur prédicteur de rappel de la longueur de la suite . Cela suggère que la mémoire humaine compresse l’information de quasi-manière optimale
  + Expérience de rappel itéré :
    - Usuellement utilisé dans l’étude de l’évolution des langues
    - Car quand on se rappelle y’a des chances qu’on modifie vite fait quelque chose et si y’a une tendance générale on peut l’observer
    - Une grille, une personne le rappel, puis une autre personne doit rappeler la grille faite précédemment
    - Résultat :
      * De plus en plus simple, surtout chez les enfants
      * La complexité baisse plus vite chez les enfants
      * Puis plus c’est simple plus on va bien retenir
    - Etudié pour les règles de grammaire
    - **Nos** **erreurs de rappel en mémoire ont tendance à aller vers plus de simplicité**
    - On peut l’interpréter comme des erreurs de compression
      * Perte d’information lors de la décompression
      * Ou juste ça ne rentre pas en MCT
  + En MCT on compresse relativement bien et proche de la compression optimale
* DS :
  + Comme les questions du début
  + Peut-être des petites démonstrations