Chapitre 4

Complexité de Kolmogorov

Le mathématicien soviétique Andreï Kolmogorov a introduit la notion de complexité dite de Kolmogorov en 1965. Il mesure la complexité d'une suite de nombre comme la taille du plus petit programme qui calcule cette suite.

Définition 4.0.1. *Soit p un programme (sans valeur d'entrée) en python et $l(p)$ le nombre de caractères qui le compose. Notons $\mathcal{O}(p)$ la valeur retournée par le programme. Soit x une chaine de caractères quelconque, on définie la complexité de Kolmogorov comme la quantité*

$$K(x) = \min_{\mathcal{O}(p)=x} l(p),$$

c'est à dire la longueur minimale d'un programme qui retourne la chaine x.

On note $\mathcal{O}(p, l(x))$ la valeur retournée par un programme pour lequel on suppose que la valeur de $l(x)$ peut être utilisée au cours de l'exécution. Cela permet de définir la complexité conditionnelle

$$K(x \mid l(x)) = \min_{\mathcal{O}(p,l(x))=x} l(p).$$

Pour donner une intuition de cette notion, considérons le programme en pseudo-code suivant composé de 58 caractère,

`Retourne les 1 123 456 789 421 832 premiers chiffres de` $\sqrt{e}$.

Dans ce langage, la complexité de Kolmogorov de la suite de nombres correspondante est majorée par 58 (fois le nombre de bits nécessaires à l'encodage de cette chaine). Tandis qu'un nombre générique de cette longueur aura une complexité proche de 1 123 456 789 421 832.

Par exemple, si on écrit

`Retourne` $x_1 x_2 \dots x_n$.

on a un code de longueur $\log x + c$ pour le nombre x, ce qui majore la complexité $K(x|l(x))$.

En fait, si on code en python, la longueur de chaque chaine de caractère est connue par construction de l'interprétation python. Pour lire le code, il faut intégrer la fin de la chaîne de caractère dans la syntaxe. C'est une subtilité qui repose sur le formalisme de la machine de Turing que nous ne verrons pas ici.

Lemme 4.0.2.

$$K(x) \leq K\left(x \mid l(x)\right) + 2\log l(x) + c.$$

Démonstration. Pour lire une chaîne de caractère en mémoire, il faut indiquer à l'interpréteur quand termine la lecture. Pour cela, on va lui faire lire dans un premier temps la longueur de la chaîne. Cela nécessitera alors $\log l(x)$. Mais il faut aussi lui dire quand s'arrêter à la lecture de ce nombre. Pour cela, on peut par exemple doubler les bits, et écrire 10 à la fin, ce qui indique la fin. On aura alors $2\log l(x)$ bits à lire. □

Noter que l'on peut itérer cette construction en écrivant la longueur des $\log l(x)$ bits avec $\log\log l(x)$ bits. En introduisant la fonction $\log^* x = \log(x) + \log\log x + \dots$ on a alors l'inégalité.

Proposition 4.0.3.

$$K(x) \leq K\left(x \mid l(x)\right) + \log^* l(x) + c.$$

Proposition 4.0.4 (Universalité de la complexité de Kolmogorov)**.** *Considérons un autre langage et notons $\mathcal{O}'(p')$ la valeur retournée par un programme p' dans ce langage. On peut définir la complexité de Kolmogorov relativement à cet autre langage*

$$K'(x) = \min_{\mathcal{O}'(p')=x} l(p').$$

Si l'on peut simuler ce nouveau langage en python, il existe une constante c telle que pour toute suite x,

$$K'(x) \leq K(x) + c.$$

Donc si chacun des deux langages peut simuler l'autre,

$$|K'(x) - K(x)| \leq c$$

La complexité est définie de manière universelle à une constante près.

Démonstration. Il suffit d'écrire le programme qui simule le nouveau langage en python, puis de l'appliquer au programme dans le nouveau langage qui retourne x. □

Remarque 4.0.5. *Dans un ordinateur, on dit qu'un langage est universel (ou Turing complet) si il peut simuler tous les autres langages de programmation. C'est évidemment le cas pour le langage assembleur et c'est aussi le cas pour tous les langages usuels (python, C, JAVA, ...).*

4.1 Exemples

On donne la complexité des trois suites suivantes

1. 0101010101010101010101010101010101010101010101010101010101010101
2. 0110101000001001111001100110011111110011101111001100100100001000
3. 1101111001110101111101101111101110101101111000101110010100111011

1.

Proposition 4.1.1. *La suite qui répète n fois* 01 *a un complexité majorée par* $\log^* n + c$.

Démonstration. On pose le programme :

`Retourne n copies de 01.`

Sa longueur est égale à une constante plus le nombre de caractères de n, soit $\lceil \log n \rceil$. □

2. Cette suite correspond au développement binaire du nombre $\sqrt{2} - 1$. Donc de même, une fois écrit le programme qui prend en entré n et retourne les n premiers bits de $\sqrt{2}$, la seule longueur variable du programme est n. Donc sa complexité est majorée par $\log^* n + c'$.

3.

Proposition 4.1.2. *Considérons x une suite aléatoire de 0 et de 1 de longueur n avec exactement k 1.*

Sa complexité est majorée par

$$K(x|l(x)) \leq \log^* k + nH\left(\frac{k}{n}\right) + c.$$

Démonstration. On peut écrire le programme suivant en pseudo-code

`Génère une boucle qui liste les suites de 0 et de 1 de longueur` n `avec` k `1 et retourne la` i`-ieme.`

Où i est le rang entre 0 et $\binom{n}{k}$ de la suite dans l'ensemble des suites de 0 et de 1 de longueur n avec k 1.

Remarquons que si on tire selon la loi de Bernoulli une suite aléatoire de longueur n avec un biais $\frac{k}{n}$, on a une probabilité de tirer une suite avec k 1 égale à

$$\binom{n}{k}\left(\frac{k}{n}\right)^k\left(1-\frac{k}{n}\right)^{n-k}$$

or c'est la répartition la plus probable avec un tel biais donc

$$1 \leq (n+1)\binom{n}{k}\left(\frac{k}{n}\right)^k\left(1-\frac{k}{n}\right)^{n-k} = (n+1)\binom{n}{k}2^{-nH(\frac{k}{n})}$$

d'où

$$\binom{n}{k} \geq \frac{2^{nH(\frac{k}{n})}}{n+1}.$$

De même

$$1 \geq \binom{n}{k}\left(\frac{k}{n}\right)^k\left(1-\frac{k}{n}\right)^{n-k} = \binom{n}{k}2^{-nH(\frac{k}{n})}$$

Donc

$$\frac{1}{n+1} \cdot 2^{nH(\frac{k}{n})} \leq \binom{n}{k} \leq 2^{nH(\frac{k}{n})}.$$

Ce qui est à mettre en parallèle avec le théorème de concentration vu plus tôt. □

Remarque 4.1.3. *En faisant une estimation plus fine de $\binom{n}{k}$ avec la formule de Stirling, on peut obtenir une meilleure majoration de la complexité par*

$$K(x|l(x)) \leq nH\left(\frac{k}{n}\right) + \frac{1}{2} \cdot \log_2 n + c$$

4.2 Lien avec l'entropie

Théoreme 4.2.1. *Considérons $\{X_i\}_{i\in\mathbb{N}}$ une suite de variables aléatoires i.i.d. à valeurs dans un ensemble fini X tirée selon une loi de probabilité notée $p(x)$ pour tout $x \in X$.*

$$\frac{1}{n} \sum_{x^n \in X^n} p(x^n)K(x^n) = E\left[\frac{1}{n}K(x^n)\right] \to H(X).$$

Démonstration. L'argument pour la borne inférieure est similaire à celui donné pour la théorie de la compression. En effet, soit $H' < H(X)$. Considérons les mots spéciaux tels que $K(x^n) < nH'$. Il ne peut y avoir plus du $2^{nH'}$ programmes de longueur nH' et donc de mots avec une telle borne de complexité. Si λ_n est la probabilité d'obtenir de tels mots, on a déjà vu par la loi faible des grands nombres que pour tout ϵ et $\eta > 0$ il faut de l'ordre de $2^{n(H(X)-\eta)}$ éléments de X^n pour que λ_n dépasse ϵ. Ce qui est bien plus petit que $2^{nH'}$ donc $\lambda_n < \epsilon$. Ainsi

$$\sum_{x^n \in X^n} p(x^n)K(x^n) \geq (1-\lambda_n)nH' + \lambda_n \geq (1-\epsilon)nH'.$$

La borne supérieure, est une conséquence de la Proposition 4.1.2. On la montre pour un alphabet binaire $X = \{0, 1\}$ dont les mots sont tirés à pile ou face avec un biais de $p \in [0, 1]$. Le cas général est similaire.

D'après la proposition mentionnée, on peut borner la complexité d'une suite par

$$K(x_1, \ldots, x_n) \leq 2\log n + nH\left(\frac{1}{n}\sum x_i\right) + c.$$

Donc

$$E\left[K(x_1, \ldots, x_n)\right] \leq 2\log n + nE\left[H\left(\frac{1}{n}\sum x_i\right)\right] + c.$$

Or, par concavité de l'entropie, $E\left[H\left(\frac{1}{n}\sum x_i\right)\right] \leq H(p)$. Donc on a bien

$$\limsup E\left[\frac{1}{n}K(x^n)\right] \leq H(p).$$

□

4.3 Algorithmiquement aléatoire

Il existe de nombreux exemples de séquences longues qui sont simples à décrire algorithmiquement, comme les premiers millions de bits de π. De la même manière, il existe aussi de grands entiers qui sont simples à décrire, tels que

$$2^{2^{2^{2^{2^2}}}} \quad \text{ou encore} \quad (100!)!$$

.

Nous montrons à présent que bien qu'il y ait des séquences simples, la plupart des séquences n'ont pas de descriptions simples. De même, la plupart des entiers ne sont pas simples. Ainsi, si nous choisissons une séquence au hasard, il est probable que nous obtenions une séquence complexe. Le théorème suivant montre que la probabilité qu'une séquence puisse être compressée de plus de k bits est plus petite que 2^{-k}.

Théoreme 4.3.1. *Soient $X_1, \ldots, X_n$ tirés selon un processus de Bernoulli de paramètre $\frac{1}{2}$. Alors*

$$P\left(K(X_1, \ldots, X_n) < n - k\right) < 2^{-k}.$$

Démonstration.

$$\begin{aligned} P\left(K(X_1, \ldots, X_n) < n - k\right) < 2^{-k} &= \#\{x_1, \ldots, x_n \mid K(x_1, \ldots, x_n) < n - k\} \cdot 2^{-n} \\ &\leq 2^{n-k} \cdot 2^{-n} \\ &= 2^{-k}. \end{aligned}$$

□

Ainsi, la plupart des séquences ont une complexité proche de leur longueur. Par exemple, la fraction de séquences de longueur n ayant une complexité inférieure à $n - 5$ est inférieure à $1/32$. Cela motive la définition suivante :

Définition 4.3.2. *Une séquence $x_1, x_2, \ldots, x_n$ est dite algorithmiquement aléatoire si*

$$K(x_1 x_2 \ldots x_n) \geq n.$$

Remarquez que, par l'argument de dénombrement, il existe, pour chaque n, au moins une séquence $x_1, \ldots, x_n$ telle que

$$K(x_1 x_2 \ldots x_n) \geq n.$$

Définition 4.3.3. *Nous disons qu'une chaîne infinie x est incompressible si*

$$\lim_{n \to \infty} \frac{K(x_1 x_2 x_3 \ldots x_n)}{n} = 1.$$

Théoreme 4.3.4 (Loi forte des grands nombres pour les séquences incompressibles)**.** *Si une chaîne $x_1, x_2, \ldots$ est incompressible, elle satisfait la loi des grands nombres au sens où*

$$\frac{1}{n} \sum_{i=1}^{n} x_i \to \frac{1}{2}.$$

Ainsi, les proportions de 0 et de 1 dans toute chaîne incompressible sont presque égales.

Démonstration. Soit $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, représentant la proportion de 1 dans $\overline{x}_n := x_1 x_2 \dots x_n$. On peut écrire un programme de longueur $\log^* n + \log^* k + nH(\theta_n) + c$ pour afficher $\overline{x}_n$. Ainsi,

$$\frac{K(\overline{x}_n)}{n} < H(\theta_n) + o(1).$$

En vertu de l'hypothèse d'incompressibilité, nous avons également la borne inférieure pour n suffisamment grand,

$$1 - \epsilon \leq \frac{K(\overline{x}_n)}{n} \leq H(\theta_n) + \epsilon.$$

Ainsi, $H(\theta_n) > 1 - 2\epsilon$. Or on sait que $H(p) = 1$ ssi $p = \frac{1}{2}$ donc par continuité de H, $\theta_n \to 1$. □

4.4 Calculabilité

Considérons l'énoncé paradoxal suivant :

`Cet énoncé est faux.`

Ce paradoxe est parfois formulé sous une forme à deux énoncés :

`Le prochain énoncé est faux. L'énoncé précédent est vrai.`

Ces paradoxes sont des versions de ce qu'on appelle le paradoxe du menteur d'Épiménide, et ils illustrent les pièges liés à l'autoréférence. En 1931, Gödel utilisa cette idée d'autoréférence pour montrer que tout système intéressant de mathématiques n'est pas complet ; il existe des énoncés dans le système qui sont vrais mais qui ne peuvent pas être prouvés dans le système. Pour y parvenir, il traduisit les théorèmes et les preuves en nombres entiers et construisit un énoncé de la forme susmentionnée, qui ne peut donc pas être prouvé vrai ou faux.

Le problème de l'arrêt en informatique est très étroitement lié au théorème d'incomplétude de Gödel. En substance, il affirme que, pour tout modèle computationnel, il n'existe pas d'algorithme général pour décider si un programme s'arrêtera ou non (continuera indéfiniment). Remarquez qu'il ne s'agit pas d'un énoncé sur un programme spécifique. De toute évidence, il existe de nombreux programmes qui peuvent facilement être démontrés comme s'arrêtant ou continuant indéfiniment. Le problème de l'arrêt dit que nous ne pouvons pas répondre à cette question pour tous les programmes. La raison en est une fois de plus l'idée d'autoréférence.

En pratique, le problème de l'arrêt peut ne pas revêtir une importance immédiate, mais il a une grande importance théorique pour définir une frontière entre ce qui peut être fait sur un ordinateur (avec une mémoire et un temps non bornés) et ce qui ne peut être fait du tout (comme prouver tous les énoncés vrais en théorie des nombres). Le théorème d'incomplétude de Gödel est l'un des résultats mathématiques les plus importants du XXe siècle, et ses conséquences sont encore en cours d'exploration. Le problème de l'arrêt est un exemple essentiel du théorème d'incomplétude de Gödel.

Proposition 4.4.1. *Il n'existe pas de fonction* `termine` *qui prend en entrée le code d'un programme python et retourne si oui ou non il va s'arrêter.*

```
def boucle():
    while True:
        pass


def paradoxe():
    if termine(paradoxe):
        boucle()

print(termine(paradoxe))
```

Démonstration. Supposons que l'on aie un tel programme, que va retourner le programme suivant ?

□

L'une des conséquences de la non-existence d'un algorithme pour le problème de l'arrêt est la non-calculabilité de la complexité de Kolmogorov. La seule façon de trouver le programme le plus court en général est d'essayer tous les programmes courts et de voir lesquels d'entre eux peuvent accomplir la tâche. Cependant, à tout moment, certains des programmes courts peuvent ne pas s'être arrêtés, et il n'y a pas de moyen effectif (mécanique fini) de dire s'ils s'arrêteront ou non et ce qu'ils afficheront. Par conséquent, il n'y a pas de moyen effectif de trouver le programme le plus court pour afficher une chaîne donnée.

La non-calculabilité de la complexité de Kolmogorov est un exemple du paradoxe de Berry. Le paradoxe de Berry demande le plus court nombre non définissable en moins de 10 mots. Un nombre comme 1 101 121 ne peut pas être une solution car l'expression elle-même est plus courte que 10 mots. Cela illustre les problèmes liés aux termes "définissable" ou "descriptible" ; ils sont trop larges pour être utilisés sans un sens strict. Si nous nous limitons à la signification "peut être décrit pour être imprimé sur un ordinateur", nous pouvons résoudre le paradoxe de Berry en disant que le plus petit nombre non descriptible en moins de 10 mots existe mais n'est pas calculable. Cette "description" n'est pas un programme pour calculer le nombre.

Comme indiqué au début du chapitre, on n'anticipe pas vraiment que les praticiens trouveront le programme informatique le plus court pour une chaîne donnée. Le programme le plus court n'est pas calculable, bien que à mesure que de plus en plus de programmes sont montrés pour produire la chaîne, les estimations par le haut de la complexité de Kolmogorov convergent vers la vraie complexité de Kolmogorov. (Le problème, bien sûr, est que l'on peut avoir trouvé le programme le plus court et ne jamais savoir qu'aucun programme plus court n'existe.) Bien que la complexité de Kolmogorov ne soit pas calculable, elle fournit un cadre pour aborder les questions de hasard et d'inférence.

4.5 Probabilités universelles

Supposons qu'un ordinateur soit nourri d'un programme aléatoire. Imaginez un singe assis à un clavier et tapant les touches au hasard. De manière équivalente, saisissez une série de lancers

de pièces équitables dans une machine de Turing universelle. Dans un cas comme dans l'autre, la plupart des chaînes de caractères ne feront pas sens pour l'ordinateur. Si une personne saisit des touches au hasard sur un terminal, elle obtiendra probablement un message d'erreur (c'est-à-dire que l'ordinateur imprimera la chaîne nulle et s'arrête). Mais avec une certaine probabilité, elle trouvera quelque chose qui a du sens. L'ordinateur imprimera alors quelque chose de signifiant. Cette séquence de sortie aura-t-elle l'air aléatoire ?

Compte tenu de nos discussions précédentes, il est clair que la plupart des séquences de longueur n ont une complexité proche de n. Puisque la probabilité d'un programme p est $2^{-l(p)}$, les programmes plus courts sont beaucoup plus probables que les plus longs ; et lorsqu'ils produisent de longues chaînes, les programmes plus courts ne produisent pas des chaînes aléatoires ; ils produisent des chaînes avec une structure décrite simplement.

La distribution de probabilité sur les chaînes de sortie est très éloignée de la distribution uniforme. Sous la distribution induite par l'ordinateur, les chaînes simples sont plus probable que les chaînes compliquées du même longueur. Cela nous motive à définir une distribution de probabilité universelle sur les chaînes comme suit :

Définition 4.5.1. *La probabilité universelle d'une chaîne x est définie par :*

$$P_{\mathcal{O}}(x) = \sum_{p:\mathcal{O}(p)=x} 2^{-l(p)} = \Pr(\mathcal{O}(p) = x).$$

Proposition 4.5.2. $\sum_x P_{\mathcal{O}}(x) \leq 1$

Démonstration. Cette somme est égale à

$$\sum_{p \text{ termine}} 2^{-l(p)}.$$

Or si un programme termine, il ne cherche pas plus loin dans le code, donc si il est préfixe d'un autre code, celui-ci ne termine pas. L'inégalité est alors une conséquence de l'inégalité de Kraft que nous verrons en TD. □

Quelle est la probabilité qu'un programme choisi au hasard en tant que séquence de lancers de pièces équitables $p_1, p_2, \ldots$ imprime la chaîne x ? Cette probabilité est universelle à bien des égards. On peut la considérer comme la probabilité d'observer une telle chaîne dans la nature ; la croyance implicite est que les chaînes plus simples sont plus probables que les chaînes compliquées. Par exemple, si nous souhaitons décrire les lois de la physique, nous pourrions considérer la chaîne la plus simple décrivant les lois comme la plus probable. Ce principe, connu sous le nom de Rasoir d'Occam, a été un principe général guidant la recherche scientifique depuis des siècles : s'il existe de nombreuses explications cohérentes avec les données observées, choisissez la plus simple. Dans notre cadre, le Rasoir d'Occam est équivalent à choisir le programme le plus court produisant une chaîne donnée.

Cette fonction de masse de probabilité est appelée universelle en raison du théorème suivant.

Théoreme 4.5.3. *Pour chaque ordinateur A,*

$$P_{\mathcal{O}}(x) \geq c' \cdot P_A(x)$$

pour chaque chaîne $x \in \{0,1\}^*$, où la constante c'_A dépend uniquement de U et A.

Démonstration. On a vu que pour chaque programme p' pour $\mathcal{O}'$ qui imprime x, il existe un programme p pour $\mathcal{O}$ de longueur au plus $l(p')+c$ produit en préfixant un programme de simulation pour $\mathcal{O}'$. Ainsi,

$$P_{\mathcal{O}}(x) = \sum_{p:\mathcal{O}(p)=x} 2^{-l(p)} \geq \sum_{p':\mathcal{O}(p')=x} 2^{-l(p')-c} = c' \cdot P_A(x).$$

□

Nous admettons que

$$P_{\mathcal{O}}(x) \approx 2^{-K(x)},$$

montrant ainsi que $K(x)$ et $\log \frac{1}{P_{\mathcal{O}}(x)}$ ont un statut équivalent en tant que mesures de la complexité algorithmique universelle. Cela est particulièrement intéressant puisque $\log \frac{1}{P_{\mathcal{O}}(x)}$ représente la longueur idéale du mot de code (la longueur du mot de code de Shannon) par rapport à la distribution de probabilité universelle $P_{\mathcal{O}}(x)$.

Nous concluons cette section avec un exemple d'un singe devant une machine à écrire par rapport à un singe devant un clavier d'ordinateur. Si le singe tape au hasard sur une machine à écrire, la probabilité qu'il tape toutes les œuvres de Shakespeare (en supposant que le texte a une longueur d'un million de bits) est de $2^{-1,000,000}$. Si le singe est assis devant un terminal d'ordinateur, cependant, la probabilité qu'il tape Shakespeare est maintenant de $2^{-K(\text{Shakespeare})} \approx 2^{-250,000}$, ce qui, bien que extrêmement faible, reste exponentiellement plus probable que lorsque le singe est devant une machine à écrire basique.

Cet exemple indique qu'une entrée aléatoire dans un ordinateur a beaucoup plus de chances de produire des sorties "intéressantes" qu'une entrée aléatoire dans une machine à écrire. Nous savons tous qu'un ordinateur est un amplificateur d'intelligence. Apparemment, il crée du sens à partir de l'absurdité également.

4.6 Paris universelles

Supposons qu'un joueur soit invité à miser de manière séquentielle sur des séquences $x \in \{0,1\}^*$. Il n'a aucune idée de l'origine de la séquence. On lui offre des cotes équitables (2 contre 1) pour chaque bit. Comment devrait-il parier ? S'il connaissait la distribution des éléments de la chaîne, il pourrait utiliser des paris proportionnels en raison de ses propriétés de croissance optimale, comme on l'a vu en TD. S'il croit que la séquence est apparue naturellement, il semble intuitif que les chaînes plus simples sont plus probables que les chaînes complexes. Ainsi, s'il devait étendre l'idée des paris proportionnels, il pourrait parier selon la probabilité universelle de la chaîne. À titre de référence, notez que si le joueur connaît la séquence x à l'avance, il peut multiplier sa richesse par un facteur de $2^{l(x)}$ simplement en pariant toute sa richesse à chaque fois sur le prochain symbole de x. Soit la richesse $S(x)$ associée au schéma de pari $b(x)$, où $\sum b(x) = 1$, donnée par

$$S(x) = 2^{l(x)} b(x).$$

Supposons que le joueur mise $b(x) = 2^{-K(x)}$ sur une séquence x. Cette stratégie de pari peut être appelée jeu universel. Notons que la somme des paris

$$\sum_x b(x) = \sum_x 2^{-K(x)} \leq \sum_{p:p \text{ s'arrête}} 2^{-l(p)} = \sum \leq 1,$$

et il n'aura pas utilisé tout son argent.

Pour simplifier, supposons qu'il jette le reste. Par exemple, le montant de richesse résultant d'un pari $b(0110)$ sur une séquence $x = 0110$ est $2^{l(x)}b(x) = 2^4 b(0110)$ plus le montant gagné sur tous les paris $b(0110...)$ sur les séquences qui prolongent x.

Nous avons le théorème suivant :

Théoreme 4.6.1. *Le logarithme de la richesse qu'un joueur atteint sur une séquence en utilisant le jeu universel, plus la complexité de la séquence, est minoré par la longueur de la séquence, soit*

$$\log S(x) + K(x) \geq l(x).$$

Remarque 4.6.2. *C'est à mettre en lien avec l'inégalité que nous avions trouvé avec la stratégie pour les courses de chevaux. C'est à dire que le multiplicateur optimal de gain est $\sum p_i \log o_i - H(p)$, où o_i sont les cotes des chevaux et p_i leur probabilité de gagner. La stratégie optimale étant de prendre des paris proportionnels à p.*

Démonstration. On calcule

$$S(x) = \sum_{x' \supseteq x} 2^{l(x)} b(x') \geq 2^{l(x)} 2^{-K(x)}$$

Où on écrit $x' \supseteq x$ si x est un préfixe de x'. □

Le résultat peut être compris de plusieurs manières. Pour des séquences infinies x avec une complexité de Kolmogorov finie,

$$S(x_1 x_2 \dots x_l) \geq 2^{l(x)-K(x)} = 2^{l(x)-c}$$

pour tout l. Puisque 2^l est le maximum qui peut être gagné dans l paris à des cotes équitables, ce schéma fonctionne asymptotiquement aussi bien que le schéma basé sur la connaissance préalable de la séquence. Par exemple, si $x = \pi_1 \pi_2 \dots \pi_n \dots$, les chiffres dans le développement de π, la richesse au temps n sera $S_n = S(x_n) \geq 2^n - c$ pour tout n.

Si la séquence est effectivement générée par un processus de Bernoulli avec un paramètre p, alors

$$S(X_1 \dots X_n) \geq 2^{n - nH(\overline{X_n}) - C \log n - c} \approx 2^{n(1 - H(p) - C\frac{\log n}{n} - \frac{c}{n})}$$

où $\overline{X_n}$ est la valeur moyenne des $X_1, \dots, X_n$. Ce qui est similaire, au premier ordre, au taux atteint lorsque le parieur connaît la distribution à l'avance, comme dans le cas de la course de chevaux.

Des exemples, nous pouvons voir que le schéma de jeu universel sur une séquence aléatoire fonctionne asymptotiquement aussi bien qu'un schéma qui utilise une connaissance préalable de la vraie distribution.

4.7 Statistique exhaustive de Kolmogorov

Supposons que nous ayons une séquence échantillon d'un processus Bernoulli(θ). Quelles sont les régularités ou déviations par rapport à l'aléatoire dans cette séquence ? Une façon de répondre à la question est de trouver la complexité de Kolmogorov d'une telle suite de longueur n, $K(x^n)$, que nous savons être approximativement $nH(\theta) + C \log n + c$. Puisque, pour $\theta \neq \frac{1}{2}$, cela est bien inférieur à n,

nous concluons que x^n a une structure et n'est pas tiré au hasard selon une distribution Bernoulli $\left(\frac{1}{2}\right)$. Mais quelle est la structure ? La première tentative pour trouver la structure est d'investiguer le programme le plus court p^* pour x^n. Cependant, la description la plus courte de p^* est d'une longueur à peu près égale à celle de p^* lui-même ; sinon, nous pourrions compresser davantage la description de x^n, contredisant la minimalité de p^*. Ainsi, cette tentative est vaine.

Un indice pour une bonne approche vient d'un examen de la manière dont p^* décrit x^n. Le programme "La séquence a k 1 ; parmi de telles séquences, c'est la ième" est optimal au premier ordre pour les séquences Bernoulli(θ). Nous notons qu'il s'agit d'une description en deux étapes, et toute la structure de la séquence est capturée dans la première étape. De plus, x^n est maximale complexe étant donné la première étape de la description. La première étape, la description de k, nécessite $\log(n+1)$ bits et définit un ensemble $S = \{x \in \{0,1\}^n : \sum x_i = k\}$. La deuxième étape nécessite $\log |S| = \log \binom{n}{k} \approx nH_0(x^n) \approx nH_0(\theta)$ bits et ne révèle rien d'extraordinaire sur x^n.

Nous reproduisons ce processus pour des séquences générales en cherchant un ensemble S simple qui contient x^n. Nous le suivons ensuite avec une description brutale de x^n dans S utilisant $\log |S|$ bits. Commençons par définir l'ensemble le plus petit contenant x^n qui est descriptible en au plus k bits.

Définition 4.7.1. *La fonction de structure de Kolmogorov $K_k(x^n)$ pour une suite $x^n \in \{0,1\}^n$ est définie par*

$$K_k(x^n) = \min_{\substack{p: l(p) \leq k \\ \mathcal{O}(p,n) = S \\ x^n \in S \subseteq \{0,1\}^n}} \log |S|.$$

La définition de l'ensemble S est le plus petit ensemble qui peut être décrit en utilisant au plus k bits et qui inclut x^n. Par $(p,n) = S$, nous entendons que l'exécution du programme p avec les données n avec le langage $\mathcal{O}$ produira la fonction indicatrice de l'ensemble S.

Définition 4.7.2. *Pour une constante c donnée, soit k^* le plus petit k tel que*

$$K_{k^*}(x^n) + k \leq K(x^n) + c.$$

Soit S^{**} l'ensemble correspondant et soit p^{**} le programme qui imprime la fonction indicatrice de S^{**}. Alors, nous dirons que p^{**} est une statistique minimale suffisante de Kolmogorov pour x^n.

Considérons les programmes p^* décrivant des ensembles S^* tels que

$$K_k(x^n) + k = K(x^n).$$

Tous les programmes p^* sont des "statistiques suffisantes" en ce sens que la complexité de x^n donnée S^* est maximale. Mais la statistique minimale suffisante est la plus courte des "statistique suffisante".

L'égalité dans la définition ci-dessus est jusqu'à une grande constante dépendante du langage de programmation. Alors k^* correspond au plus petit k pour lequel la description en deux étapes de x^n est aussi bonne que la meilleure description en une seule étape de x^n. La deuxième étape de la description fournit simplement l'indice de x^n dans l'ensemble S^{**} ; cela prend $K_k(x^n)$ bits si x^n est conditionnellement maximale complexe étant donné l'ensemble S^{**}. Ainsi, l'ensemble S^{**} capture toute la structure de x^n. La description restante de x^n dans S^{**} est essentiellement la description de l'aléatoire dans la chaîne. Par conséquent, S^{**} ou p^{**} est appelé la statistique suffisante de Kolmogorov pour x^n.

Cela est similaire à la définition d'une statistique suffisante en statistiques mathématiques. Une statistique T est dite suffisante pour un paramètre θ si la distribution de l'échantillon donné la statistique suffisante est indépendante du paramètre ; c'est-à-dire,

$$\theta \to T(X) \to X$$

forme une chaîne de Markov dans cet ordre. Pour la statistique suffisante de Kolmogorov, le programme p^{**} est suffisant pour la "structure" de la chaîne x^n ; le reste de la description de x^n est essentiellement indépendant de la "structure" de x^n. En particulier, x^n est maximale complexe étant donné S^{**}.

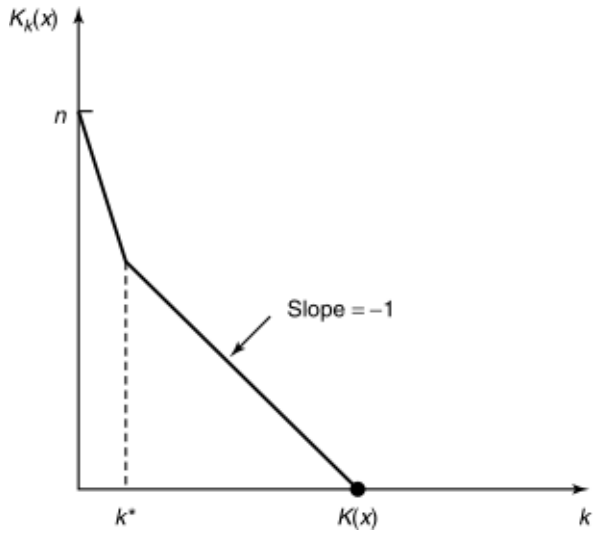

FIGURE 4.1 – Statistique exhaustive de Kolmogorov

Un graphique typique de la fonction de structure est illustré dans la Figure 14.4. Lorsque $k = 0$, le seul ensemble qui peut être décrit est l'ensemble entier $\{0,1\}^n$, de sorte que la taille correspondante de l'ensemble logarithmique soit n. En augmentant k, la taille de l'ensemble diminue rapidement jusqu'à ce que

$$k + K_k(x^n|n) \approx K(x^n|n)$$

Après cela, chaque bit supplémentaire de k réduit l'ensemble de moitié, et nous progressons le long de la ligne de pente -1 jusqu'à $k = K(x^n|n)$. Pour $k \geq K(x^n|n)$, le plus petit ensemble qui peut être décrit et qui inclut x^n est le singleton $\{x^n\}$, et donc $K_k(x^n|n) = 0$.

Illustrons maintenant le concept avec quelques exemples.

1. Séquence Bernoulli(θ). Considérons un échantillon de longueur n d'une séquence Bernoulli avec un paramètre inconnu θ. Nous pouvons décrire cette séquence avec $nH(\frac{k}{n}) + \frac{1}{2} \cdot \log n + c$ bits en utilisant une description en deux étapes où nous décrivons k dans la première étape

(en utilisant $\log n$ bits) puis décrivons la séquence dans toutes les séquences avec k uns (en utilisant $\log \binom{n}{k}$ bits). Cependant, nous pouvons utiliser une description en première étape encore plus courte. Au lieu de décrire k exactement, il est possible de diviser la plage des k possibles en intervalles et décrivons k uniquement avec une précision de

$$\sqrt{\frac{k}{n} \cdot \frac{n-k}{n}} \sqrt{n}$$

en utilisant $\frac{1}{2} \log n$ bits.

Ensuite, nous décrivons la séquence réelle parmi toutes les séquences dont le type est dans le même intervalle que k. La taille de l'ensemble de toutes les séquences avec l uns, $l \in k \pm \sqrt{\frac{k}{n} \cdot \frac{n-k}{n}} \sqrt{n}$ est $nH_{\frac{k}{n}} + o(n)$ selon la formule de Stirling, de sorte que la longueur totale de la description est toujours $nH_{\frac{k}{n}} + \frac{1}{2} \log n + o(n)$, mais la longueur de description de la statistique suffisante de Kolmogorov est $k^* \approx \frac{1}{2} \log n$.

2. Échantillon d'une chaîne de Markov. Dans la même veine que l'exemple précédent, considérons un échantillon d'une chaîne binaire de Markov d'ordre un. Dans ce cas également, p^{**} correspondra à la description du type de Markov de la séquence (le nombre d'occurrences de 00, 01, 10 et 11 dans la séquence) ; cela transmet toute la structure de la séquence. Le reste de la description sera l'indice de la séquence dans l'ensemble de toutes les séquences de ce type de Markov. Ainsi, dans ce cas, $k^* \approx 2(\frac{1}{2} \log n) = \log n$, correspondant à la description de deux éléments du type joint conditionnel avec une précision appropriée. (Les autres éléments du type joint conditionnel peuvent être déterminés à partir de ceux-ci.)