# Complexité de Kolmogorov

8 janvier 2017

## 1 Introduction

On croise souvent des problèmes-jeux du type « compléter la suite 2, 4, 8, ... ». D'un point de vue mathématique, on peut être tenté de penser que tout nombre $x$ conviendrait également bien, puisqu'il est aisé de trouver une fonction qui interpole la suite et passe par $x$.
Toutefois, la notion de complexité de Kolmogorov permet de donner une interprétation rigoureuse au concept de méthode la plus simple pour calculer une suite. En première approximation, on peut définir la complexité de Kolmogorov $K(u)$ d'une suite $u$ comme la taille minimale d'un programme (encodé en binaire) qui calcule cette suite, dans un langage donné (qu'il s'agisse de lambda-calcul ou de Python).

Dans toute la suite, $u$ désigne une suite finie de 0 et de 1.

## 2 Définitions

La définition de la complexité de Kolmogorov en partant d'un langage donné est une bonne approche, mais le choix du langage est trop arbitraire pour être satisfaisant. Toutefois, ce n'est pas réellement un problème, car les complexités définies à partir de deux langages A et B sont très liées : Il suffit d'écrire un interpréteur de B dans le langage A (pour peu qu'il soit Turing-complet) et on peut alors établir l'existence d'une constante $c$ telle que

$$\forall u, K_A(u) \leq K_B(u) + c.$$

Essayons maintenant d'établir la définition la plus générique possible :
Soit $f$ une fonction partielle calculable sur l'ensemble des chaînes de bits. On définit alors la complexité de Kolmogorov $K_f$ par

$$K_f(u) = \min\{|y| \quad | \quad f(y) = x\}$$

On se rend compte que la définition préalable à l'aide d'un langage de programmation est valable : la fonction qui à un programme associe le résultat de son évaluation est calculable. On peut alors établir notre premier théorème, qui stipule que la définition à partir d'un langage de programmation Turing-complet est en réalité la bonne définition :

**Théorème 1.** *Il existe une fonction $f$ calculable telle que pour toute fonction $g$ calculable, on aie l'existence d'une constante $c_g$ qui vérifie $\forall u, K_f(u) \leq K_g(u) + c_g$. On notera dorénavant $K$ pour $K_f$.*

*Démonstration.* Il suffit pour s'en convaincre de choisir pour $f$, comme précédemment, l'évaluation d'un langage Turing-complet. On écrit alors un algorithme qui calcule $g$ dans notre langage (puisque $g$ est calculable), et sa taille donne la constante $c_g$. □

La complexité de Kolmogorov $K(u)$ est alors définie à une constante près, qui est indépendante de $u$. Dans la suite, on utilisera les notations de Landau pour représenter de telles constantes.

# 3 Observations élémentaires

Tout d'abord, quelques majorations naturelles de la complexité de Kolmogorov :

**Théorème 2.** *1. $K(u) \leq |u| + O(1)$.*

*2. Si $f$ est une fonction calculable, $K(f(u)) \leq K(u) + O(1)$.*

*Démonstration.* 1. La fonction identité $Id$ est calculable, et $K_{Id}(u) = |u|$.

2. Pour s'en convaincre, il suffit d'écrire $f$ dans le langage choisi et de prendre le programme qui l'applique à $u$.

□

On peut alors se demander si ces majorations (et en particulier $K(u) \leq |u|+O(1)$) sont efficaces. La réponse est oui, par de simples arguments de comptage : en effet, on peut décrire au plus $2^{n+1}-1$ suites à l'aide d'un programme de taille au plus $n$. (On rappelle que le programme est encodé en binaire dans notre exposé.)

**Théorème 3.** *1. Le nombre de suites $u$ telles que $K(u) \leq n$ est égal à $\Theta(2^n)$.*

*2. Il existe une constante $c$ telle que quel que soit $n$, 99% des suites de taille $n$ aient une complexité de Kolmogorov comprise entre $n - c$ et $n + c$.*

*Démonstration.* 1. On établit que ce nombre est plus petit que $2 \cdot 2^n$ comme écrit ci-dessus. Si $c$ vérifie pour toute suite $u$, $K(u) \leq |u| + c$, alors on obtient une minoration du nombre de suites de complexité $\leq n$ par $2^{-c} \cdot 2^n$. D'où le résultat.

2. Si on prend $c \geq 7$, alors on a au plus $2^n/2^7$ suites de complexité $\leq n - 7$. Donc comme il y a $2^n$ suites de taille $n$, on en déduit qu'au moins 99% ont une complexité $\geq n - 7$. La constante pour la majoration est donnée par $K(u) \leq |u| + O(1)$.

□

Maintenant qu'on a obtenu une bonne estimation de la complexité de Kolmogorov pour l'immense majorité des suites, on peut chercher à la calculer effectivement. Dans ce but, une première observation est que $K$ est « énumérable par valeurs supérieures », c'est à dire qu'il existe une fonction calculable totale $k$ qui vérifie pour toute suite $u$ :

$$k(u, 0) \geq k(u, 1) \geq k(u, 2) \geq \dots$$

$$\lim_{n \to \infty} k(u, n) = K(u)$$

Pour s'en convaincre, on définit $k(u, n)$ comme le programme le plus court qui calcule $u$ en au plus $n$ étapes. S'il n'en existe pas de taille $\leq |u|+c$, alors on pose $k(u, n) = |u|+c$ (où $c$ est la constante telle que $K(u) \leq |u| + c$). Il est clair que $k$ est totale et calculable, et qu'elle tend vers $K$.

Notons toutefois que cette méthode ne fournit pas d'algorithme qui calcule $K$. En effet, on ne donne pas de méthode qui permet de se rendre compte du $n$ à partir duquel $k(u, n) = K(u)$. En réalité, la complexité de Kolmogorov n'est pas calculable !

**Théorème 4.** *La complexité de Kolmogorov $K$ n'est pas calculable. Plus généralement, toute fonction calculable $f$ qui vérifie $f(u) \leq K(u)$ est bornée.*

*Démonstration.* Supposons qu'il existe une fonction $f$ calculable non bornée qui vérifie $f(u) \leq K(u)$. Alors on peut construire une fonction calculable $g$ définie sur les entiers qui vérifie $K(g(n)) \geq n$ (en effet, pour calculer $g(n)$ on énumère les chaînes jusqu'à en trouver $u_n$ telle que $f(u_n) \geq n$). Mais alors on a aussi, comme démontré précédemment, $K(g(n)) \leq K(n)+O(1)$. La représentation binaire de $n$ permet d'obtenir que $K(n) \leq log(n) + O(1)$. On obtient alors $n \leq log(n) + O(1)$, ce qui est contradictoire. □

# 4 Liens avec l'incomplétude

La complexité de Kolmogorov est en fait très étroitement liée aux théorèmes d'incomplétudes des théories mathématiques. Cela provient du fait qu'une preuve mathématique (dans une théorie récursivement axiomatisable et cohérente) peut être, sous un certain angle, vue comme programme informatique.

Le premier résultat de cette section est une sorte d'extension de la non-calculabilité de bornes inférieures pour la complexité de Kolmogorov au langage des mathématiques :

**Théorème 5.** *Il existe une constante $c$ telle que tous les théorèmes prouvables de la forme « $K(u) \geq n$ » vérifient $n \leq c$.*

*Démonstration.* Supposons le contraire. Alors on peut construire une fonction calculable $f$ définie sur les entiers de la manière suivante : pour calculer $f(n)$, on énumère les preuves mathématiques jusqu'à tomber sur une preuve valide d'un théorème du type $K(u) \geq n$, et on pose alors $f(n) = u$. Cette fonction est bien calculable si on suppose notre théorie récursivement axiomatisable, et doit donc vérifier $K(f(n)) \leq log(n) + O(1)$, ce qui est absurde si elle est de plus cohérente. □

Ceci peut paraître très surprenant lorsqu'on peut démontrer que presque toutes les suites de taille $n$ ont une complexité de Kolmogorov avoisinant $n$. On obtient ainsi une famille de théorèmes vrais mais indémontrables.

Les liens entre complexité de Kolmogorov et incomplétude se poursuivent encore bien plus profondément, citons par exemple un résultat récent de Laurent Bienvenu (voir [3]) :

**Théorème 6.** *Rajouter à une théorie une quantité dénombrable d'axiomes du type « $K(u) \geq n$ » suffit à rendre toute formule $\Pi_1$ décidable. (Une formule $\Pi_1$ est de la forme $\forall x, \phi(x)$ avec $\phi$ sans quantificateurs.)*

Ainsi, la complexité de Kolmogorov suffit à concentrer toute l'indécidabilité des formules $\Pi_1$. Notons que la preuve du théorème précédent ne s'applique nécessairement plus à une théorie ainsi modifiée. Cela signifie donc qu'une telle théorie n'est plus récursivement axiomatisable.

# 5 Construction supplémentaires

Avant de continuer à propos des suites de complexité élevée, faisons un aparté le temps de définir deux notions qui nous serons utiles par la suite.

La première des deux est la complexité d'une paire. Pour la définir, choisissons une injection calculable des paires de chaînes dans les chaînes, que l'on notera $(u, v) \to [u, v]$. On définit alors la complexité d'une paire par $K(u, v) = K([u, v])$. Comme la fonction d'appariement est calculable, on démontre aisément qu'un choix différent donnera une complexité de paire qui ne diffère que d'une constante.

La seconde est la complexité conditionnelle. Elle sert à mesurer la complexité d'une suite en en connaissant une seconde. On définit donc la complexité $K(u|v)$ de $u$ sachant $v$ comme la taille minimale d'un programme qui prend $v$ en entrée et renvoie $u$.
On peut établir pour cette construction des résultats élémentaires analogues à ceux de la complexité de Kolmogorov. (Notamment l'indépendance au langage, au prix d'une constante.)

# 6 Incompressibilité

Nous avons donc vu qu'il existe de nombreuses suites de complexité de Kolmogorov sensiblement égale à leur taille, mais qu'on ne peut pas en exhiber une à partir d'une certaine taille. Examinons plus en détail les propriétés de ces suites dites « incompressibles ».

On dit qu'une chaîne $u$ de taille $n$ est $c$-incompressible lorsqu'elle vérifie $K(u|n) \geq n - c$. Le théorème suivant formule, toujours avec les mêmes arguments de comptage, que la majorité des suites sont $c$-incompressibles :

**Théorème 7.** *Pour tout $n$, la proportion de chaînes $c$-incompressibles de taille $n$ est strictement supérieure à $1 - 2^{-c}$.*

*Démonstration.* La quantité de chaînes qu'il est possible de décrire avec $n-c-1$ bits est $2^{n-c}-1$. Donc la proportion de chaînes de longueur $n$ qui ne sont pas $c$-incompressibles est strictement inférieure à $2^n/2^{n-c}$. D'où le résultat. □

On en déduit en particulier qu'il existe, pour tout $n$, au moins une chaîne de taille $n$ qui est 0-incompressible (que l'on abrège en incompressible).

Manipuler des chaînes incompressibles permet d'obtenir de nombreux résultats en utilisant le fait qu'il existe des objets qui ne présentent « aucune régularité ». Les quelques exemples suivants montrent des applications diverses de ce principe.

**Théorème 8.** *Il existe une infinité de nombres premiers.*

*Démonstration.* Si ce n'est pas le cas, alors tout nombre $n$ peut s'écrire

$$n = p_1^{n_1} \cdot p_2^{n_2} \cdot \ldots \cdot p_m^{n_m}$$

On remarque aisément que pour tout $k$, $n_k \leq log(n)$. Donc pour coder $n$, il suffit de coder la suite $n_1, \ldots, n_m$, ce qui nécessite $O(m \cdot log(log(n)))$ bits. Cela contredit l'existence de suites (donc de nombres, une fois écrits en binaire) incompressibles. □

**Théorème 9.** *Soit $M$ une machine de Turing. Alors il existe une constante $c$ qui vérifie la propriété suivante :*
*Pour tout $k$, pour tout $l \geq k$, et pour tout $t$, si la bande est initialement vide à partir de la case $k$, alors après $t$ étapes, la complexité de la chaîne écrite sur la bande à partir de la case $l$ est au plus*

$$c \cdot t/(l-k) + O(log(l) + log(t)).$$

*Démonstration.* On a envie de coder la chaîne résultante à l'aide de la machine et de son entrée. Toutefois, on prendra garde à ne pas utiliser l'entrée en entier : en effet, seule l'information effectivement transportée est à prendre en compte. On stocke donc l'information que l'on voit transiter, ainsi qu'une description de $M$, la valeur de $l$ et des marqueurs temporels pour pouvoir reconstruire le comportement de $M$ sur la partie de la bande après $l$. On obtient la borne souhaitée avec un codage adapté. □

On peut alors utiliser ce théorème couplé à l'incompressiblité pour déterminer qu'une machine de Turing qui duplique l'entrée a besoin de $\Omega(n^2)$ étapes pour une entrée de taille $n$.

**Théorème 10.** *Une matrice dans $\mathcal{M}_n(\mathbb{F}_2)$ incompressible a un rang d'au moins $n/2$. De plus, elle a des mineurs de rang élevé (dans un sens qu'on ne définit pas ici, cf [2]).*

L'idée de la preuve est là encore de supposer le contraire, et d'en déduire un stockage efficace de la matrice. Si il est trivial de construire une matrice de rang élevé, construire une matrice dont tous les mineurs ont un rang élevé est beaucoup plus complexe, et nécessaire dans certaines preuves.

**Théorème 11.** *Un graphe incompressible est connexe.*

(Un graphe de $n$ sommet revient à une chaîne binaire de $n(n-1)/2$ bits.)

# 7 Liens avec la notion d'aléatoire

Pour se rendre compte du lien entre la complexité de Kolmogorov et l'aléatoire, il suffit de générer un fichier dont chaque bit est aléatoire et d'essayer de le compresser à l'aide de n'importe quel logiciel prévu à cet effet : sa taille ne diminue pas.
Cela est simplement expliqué par le fait que 99.9% des chaînes de taille $n$ sont 10-incompressibles (par exemple). On obtient donc une idée raisonnable de le la définition que devrait avoir une suite

aléatoire infinie : ne pas posséder de régularité, porter un maximum d'information.

Commençons par quelques éléments de probabilités :
Soit $\Omega$ l'ensemble des suites binaires infinies, et pour toute suite finie $u$, on note $\Omega_u$ l'ensemble des suites qui commencent par $u$. On équipe $\Omega$ de la mesure de probabilité $\mathbb{P}$ qui est déterminée par $\mathbb{P}(\Omega_u) = 2^{-|u|}$ pour tout $u$. (La tribu est ici celle engendrée par les $\Omega_u$.) Une partie $X$ de $\Omega$ est dite récursivement négligeable [1] si il existe un algorithme (dit de recouvrement) qui prend un nombre $\epsilon > 0$ en entrée, et renvoie une suite $x_{\epsilon,0}, x_{\epsilon,1}, \ldots$ telle que

$$X \in \Omega_{x_{\epsilon,0}} \cup \Omega_{x_{\epsilon,1}} \cup \ldots$$

$$\sum_k 2^{-|x_{\epsilon,k}|} \leq \epsilon.$$

Les ensembles récursivement négligeables représentent ceux que l'on peut « isoler » du reste. Quelques remarques élémentaires :

**Théorème 12.** *1. Un singleton calculable est récursivement négligeable.*

*2. Un sous-ensemble d'un ensemble récursivement négligeable est récursivement négligeable.*

*3. Une union finie d'ensemble récursivement négligeables est récursivement négligeable.*

*4. Soit $X_0, X_1, \ldots$ une famille d'ensembles récursivement négligeables telle qu'il existe un algorithme qui lorsqu'on lui entre $n$ renvoie un algorithme de recouvrement pour $X_n$. Alors l'union de la famille est récursivement négligeable.*

*Démonstration.* 1. Il suffit de prendre un algorithme qui, lorsqu'on lui entre $\epsilon$, calcule les $-log(\epsilon)$ premières décimales.

2. Évident.

3. Pour une union de deux ensembles récursivement énumérables, on fait tourner les deux algorithmes de recouvrement avec pour entrée $\epsilon/2$ et on prend l'union de leurs sorties. On généralise à une union finie.

4. On fait tourner l'algorithme $n$ avec pour entrée $2^{-n}\epsilon$ et on prend l'union de toutes les sorties (en « diagonale »).

□

Une remarque cruciale est l'existence d'un ensemble récursivement négligeable maximal. Il existe une infinité indénombrable d'ensembles récursivement négligeables (puisqu'ils sont stables par parties et qu'il en existe un infini — par exemple l'ensemble des suites dont tout terme de rang pair vaut 0), donc il est déraisonnable de chercher à prendre leur union. L'astuce employée ici est de prendre l'union sur tous les algorithmes de recouvrement possibles qui, eux, sont dénombrables.

**Théorème 13.** *L'union de tous les ensembles récursivement négligeables est récursivement négligeable.*

*Démonstration.* On voudrait donc énumérer tous les algorithmes de recouvrement. Toutefois, il est difficile de déterminer si un algorithme est effectivement un algorithme de recouvrement. Pour pallier cette difficulté, on va transformer les algorithmes : si un algorithme renvoie une séquence $x_0, x_1, \ldots$ telle que $\sum_{k=0}^{n} 2^{-|x_k|} > \epsilon$, alors on retire $x_n$ de la suite. Tous les algorithmes deviennent des algorithmes de recouvrement une fois transformés — et les algorithmes de recouvrement sont laissés invariants. On peut donc prendre l'union des ensembles recouverts. □

On définit alors une suite comme étant aléatoire (au sens de Martin-Löf) si elle n'appartient pas à l'ensemble récursivement négligeable maximal. On observe immédiatement que la mesure de l'ensemble des suites aléatoires est 1.

1. Traduction courageuse de *effectively null*

**Théorème 14.** *Si $u$ est une suite infinie aléatoire, alors*

$$\lim_{n\to\infty} \frac{u_0 + u_1 + \ldots + u_{n-1}}{n} = \frac{1}{2}$$

La loi forte des grands nombres établit que l'ensemble des suites qui n'ont pas pour moyenne $1/2$ est négligeable. L'idée est alors de montrer que les procédés utilisés dans la preuve de la loi forte des grands nombres sont calculables, et que cet ensemble est donc récursivement négligeable.

Soit $S$ une fonction totale calculable des chaînes de bits dans $\{0,1\}$. Si $u$ est une suite binaire infinie, la sous-suite sélectionnée par $S$ est la sous-suite formée par les $u_n$ tels que $S(u_0u_1\ldots u_{n-1}) = 1$.

**Théorème 15.** *Une sous-suite d'une suite aléatoire sélectionnée par une fonction calculable est soit finie, soit aléatoire.*

On en déduit qu'une sous-suite aléatoire sélectionnée par une fonction calculable a pour valeur moyenne $1/2$. Ainsi, lorsqu'une suite aléatoire est énoncée, on ne peut pas prévoir raisonnablement la valeur des bits futurs.

# 8 Algorithmes probabilistes

On peut maintenant se demander comment la notion de complexité de Kolmogorov est modifiée si on autorise nos programmes à ne plus être déterministes — plus spécifiquement, à accéder à une variable aléatoire suivant une loi de Bernouilli (0 ou 1 avec probabilité $1/2$). Un algorithme ne renvoie alors plus une unique chaîne, mais on peut considérer la distribution de probabilité induite sur l'ensemble des chaînes possibles.

Un nombre réel $p$ est dit calculable par valeurs inférieures si il existe un algorithme qui prend un entier $n$ en entrée et renvoie un réel $p_n$, et qui vérifie

$$p_0 \leq p_1 \leq \ldots$$

$$p = \lim_{n\to\infty} p_n$$

Une suite de nombre réels est dite calculable par valeurs inférieures si chaque terme l'est et s'il existe un algorithme qui, lorsqu'on lui entre $n$, renvoie l'algorithme de calcul pour le $n$-ième terme. Le théorème suivant indique que les distributions de probabilité calculables par valeurs inférieures sont exactement les distributions qui peuvent être obtenues à partir d'un programme non déterministe.

**Théorème 16.**
1. *Si $A$ est un algorithme non déterministe dont les sorties sont des nombres entiers, et si on note $p_n$ la probabilité de l'évènement « $A$ renvoie $n$ », alors $(p_n)_{n\in\mathbb{N}}$ est calculable par valeurs inférieures et $\sum_n p_n \leq 1$. On dit que $p$ est une semi-mesure calculable par valeurs inférieures.*
2. *Si $(p_n)_{n\in\mathbb{N}}$ est une semi-mesure calculable par valeurs inférieures, alors il existe un algorithme non déterministe qui renvoie $n$ avec probabilité $p_n$.*

On note qu'on ne demande que $\sum_n p_n \leq 1$ et non $\sum_n p_n = 1$. En effet, le programme peut ne pas terminer avec une probabilité non nulle, et $1 - \sum_n p_n$ correspond donc à la probabilité qu'il ne termine pas.
On a donc une caractérisation des semi-mesures calculable par valeurs inférieures à l'aide d'algorithmes. Comme ces derniers sont en quantité dénombrable, on peut en déduire l'existence d'une semi-mesure calculable par valeurs inférieures maximales — à l'instar de ce qui a été fait pour la complexité de Kolmogorov ou les ensembles récursivement négligeables.

**Théorème 17.** *Il existe une semi-mesure calculable par valeurs inférieures $\mu$ maximale, c'est à dire que pour toute autre semi-mesure calculable par valeurs inférieures $p$, il existe une constante $c_p$ telle que*

$$\forall n \in \mathbb{N}, p(n) \leq c_p \cdot \mu(n).$$

*Démonstration.* Choisir une semi-mesure calculable par valeurs inférieures revient à choisir un algorithme non déterministe. On commence par choisir une mesure de probabilité $\mathbb{P}$ sur $\mathbb{N}$, telle que $\forall n, \mathbb{P}(n) > 0$, ainsi qu'une énumération des algorithmes non déterministes. On construit alors pour déterminer $\mu$ un algorithme qui choisit le $n$-ième algorithme avec probabilité $\mathbb{P}(n)$, et qui le simule.
Si $A$ est un algorithme correspondant à $p$, il apparait en position $n_A$ dans la liste. Donc avec probabilité $\mathbb{P}(n_A)$, on va simuler cet algorithme. D'où $p \leq \frac{1}{\mathbb{P}(n_A)}\mu$. □

Une telle mesure est appelée probabilité *a priori*. En effet, n'importe quelle probabilité calculable (dans le sens discuté ci-dessus) peut s'y ramener.

# 9 Complexité préfixe

La notion de probabilité *a priori* est fortement liée à une mesure de complexité particulière, qu'on appellera complexité préfixe.
La complexité de Kolmogorov peut s'interpréter comme une compression maximale de l'information. La complexité préfixe revient, elle, à la méthode de compression optimale telle qu'aucune suite, une fois compressée, ne soit préfixe d'une autre compression.
Une fonction $f$ partielle calculable est dite préfixe si pour toutes chaînes de bits $x$ et $y$, $f(x)$ est préfixe de $f(y)$ si et seulement si $x = y$. On prouve alors qu'il existe une fonction préfixe $g$ telle que si $f$ est une fonction préfixe, alors

$$\forall u, K_g(u) \leq K_f(u) + O(1).$$

On note alors $KP$ pour $K_g$ et on l'appelle complexité préfixe.
On sait que la complexité de Kolmogorov vérifie $K(u) \leq |u| + O(1)$. Mais l'identité, qui a été utilisée pour prouver ce résultat, n'est pas une fonction préfixe. En réalité, ceci n'est pas vrai pour la complexité préfixe, comme le montre le théorème suivant :

**Théorème 18.**

$$\sum_u 2^{-|KP(u)|} \leq 1.$$

*Démonstration.* Soit $g$ une fonction préfixe qui vérifie $K_g = KP + O(1)$, et soit $G$ définie par $G(u) = argmin\{|v| \quad | \quad g(v) = u\}$. Les ensembles $\Omega_{G(u)}$ des suites infinies commençant par $G(u)$ sont disjoints (par définition de la complexité préfixe). Comme discuté précédemment, il existe une mesure de probabilité $\mathbb{P}$ sur $\Omega$ telle que $\mathbb{P}(\Omega_u) = 2^{-|u|}$. On en déduit que $\sum_u \mathbb{P}(\Omega_{G(u)}) \leq \mathbb{P}(\Omega)$ ce qui donne le résultat annoncé. □

Si l'on ne peut pas obtenir $KP(u) \leq |u| + O(1)$, il existe tout de même des majorations satisfaisantes de la complexité préfixe :

**Théorème 19.**

$$KP(u) \leq 2|u| + O(1)$$
$$KP(u) \leq |u| + 2log(|u|) + O(1)$$
$$KP(u) \leq |u| + log(|u|) + 2log(log(u)) + O(1)$$

*et ainsi de suite.*

Explicitons maintenant le lien annoncé entre complexité préfixe et probabilité *a priori* :

**Théorème 20.** *Soit $\mu$ mesure de probabilité* a priori. *Alors*

$$KP(u) = -log(\mu(u)) + O(1).$$

Ainsi, la difficulté à produire une suite à l'aide d'un programme non déterministe est fortement liée à la difficulté à la décrire de manière préfixe. Terminons notre exposé par un lien entre la complexité préfixe et les suites aléatoires au sens de Martin-Löf :

**Théorème 21.** *Une suite infinie $u_0u_1\ldots$ est aléatoire au sens de Martin-Löf si et seulement si il existe une consante $c$ telle que pour tout $n$, on aie*

$$KP(x_0x_1\ldots x_n) \geq n - c.$$

# 10 Conclusion

Ce rapide exposé de la complexité de Kolmogorov est en quasi-totalité extrait de [1]. Les preuves qui ont ici été omises ou schématisées par souci de concision peuvent y être trouvées dans leur intégralité. Le lecteur particulièrement intéressé par le sujet trouvera un exposé plus complet dans [2].

# Références


[1] Alexander Shen (2007). *Algorithmic Information Theory and Kolmogorov Complexity*

[2] Ming Li, Paul Vitanyi (2008). *An Introduction to Kolmogorov Complexity and Its Applications*, Springer

[3] Laurent Bienvenu, Andrei Romashchenko, Alexander Shen, Antoine Taveneaux, Stijn Vermeeren (2013). *The axiomatic power of Kolmogorov complexity*