Lemme local de Lovász, k-SAT et complexité de Kolmogorov

Keno Merckx

Le lemme de 1975

Théorème (Erdős-Lovász)

Soient $A_1, \ldots, A_n$ des évènements (typiquement mauvais) tels que, pour tout i:

- $\Pr[A_i] \leq p$,
- *A_i est mutuellement indépendant de tous les événements sauf au plus d autres.*

Si $4pd \leq 1$, alors, avec une probabilité positive, aucun des A_i ne se produit.

Le lemme de 1975

Théorème (Erdős-Lovász)

Soient $A_1, \ldots, A_n$ des évènements (typiquement mauvais) tels que, pour tout i:

- $\Pr[A_i] \leq p$,
- *A_i est mutuellement indépendant de tous les événements sauf au plus d autres.*

Si $4pd \leq 1$, alors, avec une probabilité positive, aucun des A_i ne se produit.

$\Rightarrow$ Il existe une version aux hypothèses plus faibles, où la condition $4pd \leq 1$ est remplacée par $ep(d+1) \leq 1$.

Le lemme de 1975

Théorème (Erdős-Lovász)

Soient $A_1, \ldots, A_n$ des évènements (typiquement mauvais) tels que, pour tout i:

- $\Pr[A_i] \leq p$,
- *A_i est mutuellement indépendant de tous les événements sauf au plus d autres.*

Si $4pd \leq 1$, alors, avec une probabilité positive, aucun des A_i ne se produit.

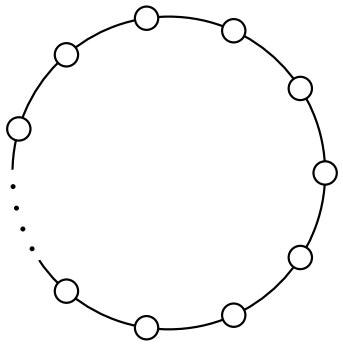
$\Rightarrow$ Il existe une version aux hypothèses plus faibles, où la condition $4pd \leq 1$ est remplacée par $ep(d+1) \leq 1$.

Différentes versions existent:

- Lemme local généralisé,
- Lemme local asymétrique,
- Lemme local pondéré,
- Lemme local inégal.

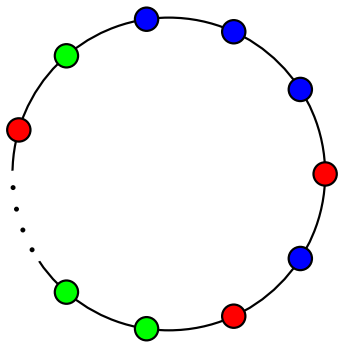
Exemple d'utilisation du LLL

Soient $11n$ points sur un cercle et un ensemble de n couleurs.



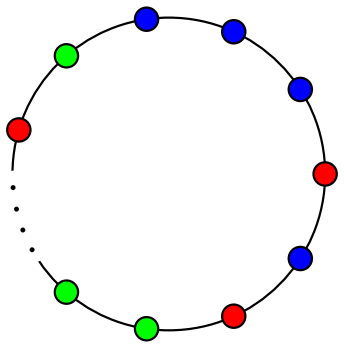
Exemple d'utilisation du LLL

Soient $11n$ points sur un cercle et un ensemble de n couleurs. Colorions les points en utilisant chaque couleur exactement 11 fois.



Exemple d'utilisation du LLL

Soient $11n$ points sur un cercle et un ensemble de n couleurs. Colorions les points en utilisant chaque couleur exactement 11 fois.

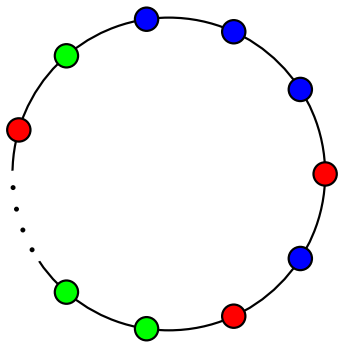


Proposition

Pour toute coloration, il existe un ensemble E de n points de couleur différente tel que deux points de E ne soient jamais adjacents sur le cercle.

Exemple d'utilisation du LLL

Soient $11n$ points sur un cercle et un ensemble de n couleurs. Colorions les points en utilisant chaque couleur exactement 11 fois.



Proposition

Pour toute coloration, il existe un ensemble E de n points de couleur différente tel que deux points de E ne soient jamais adjacents sur le cercle.

Montrons cela avec le LLL. Supposons que nous choisissons un point de chaque couleur de façon aléatoire.

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,
- la probabilité que l'évènement A_i (pour tout $1 \leq i \leq 11n$) arrive est au plus de $\frac{1}{11^2}$,

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,
- la probabilité que l'évènement A_i (pour tout $1 \leq i \leq 11n$) arrive est au plus de $\frac{1}{11^2}$,
- le choix d'une paire $\{a, b\}$ ne dépend que des paires $\{x, y\}$ qui partagent une couleur avec a ou avec b, il existe dans le pire des cas, 42 paires de type $\{x, y\}$.

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,
- la probabilité que l'évènement A_i (pour tout $1 \leq i \leq 11n$) arrive est au plus de $\frac{1}{11^2}$,
- le choix d'une paire $\{a, b\}$ ne dépend que des paires $\{x, y\}$ qui partagent une couleur avec a ou avec b, il existe dans le pire des cas, 42 paires de type $\{x, y\}$.

Appliquons le lemme local de Lovász avec les évènements A_i, $p = \frac{1}{121}$ et $d = 42$.

$$ep(d+1) \approx 0,966 < 1$$

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,
- la probabilité que l'évènement A_i (pour tout $1 \leq i \leq 11n$) arrive est au plus de $\frac{1}{11^2}$,
- le choix d'une paire $\{a, b\}$ ne dépend que des paires $\{x, y\}$ qui partagent une couleur avec a ou avec b, il existe dans le pire des cas, 42 paires de type $\{x, y\}$.

Appliquons le lemme local de Lovász avec les évènements A_i, $p = \frac{1}{121}$ et $d = 42$.

$$ep(d + 1) \approx 0,966 < 1$$

Et donc, il y a une probabilité positive que aucun des A_i ne se réalisent, ce qui veut dire que l'ensemble E de la proposition précédente doit exister.

Exemple d'utilisation du LLL (2)

Nous avons donc:

- les événements A_i qui représentent le choix d'une paire de points $\{a, b\}$ dans E, avec a adjacent à b,
- la probabilité que l'évènement A_i (pour tout $1 \leq i \leq 11n$) arrive est au plus de $\frac{1}{11^2}$,
- le choix d'une paire $\{a, b\}$ ne dépend que des paires $\{x, y\}$ qui partagent une couleur avec a ou avec b, il existe dans le pire des cas, 42 paires de type $\{x, y\}$.

Appliquons le lemme local de Lovász avec les évènements A_i, $p = \frac{1}{121}$ et $d = 42$.

$$ep(d+1) \approx 0,966 < 1$$

Et donc, il y a une probabilité positive que aucun des A_i ne se réalisent, ce qui veut dire que l'ensemble E de la proposition précédente doit exister.

$\Rightarrow$ Preuve non constructive.

Le problème k-SAT

Petit rappel:

Problème (k-SAT)

Etant donné ϕ, une formule booléenne k-CFN, décider s'il existe une assignation qui satisfait ϕ.

Le problème k-SAT

Petit rappel:

Problème (k-SAT)

Etant donné ϕ, une formule booléenne k-CFN, décider s'il existe une assignation qui satisfait ϕ.

- Pour $k = 2$, le problème est facile (dans $\mathcal{P}$).
- Pour $k \geq 3$, moins facile ($\mathcal{NP}$-complet).

Le problème k-SAT

Petit rappel:

Problème (k-SAT)

Etant donné ϕ, une formule booléenne k-CFN, décider s'il existe une assignation qui satisfait ϕ.

- Pour $k = 2$, le problème est facile (dans $\mathcal{P}$).
- Pour $k \geq 3$, moins facile ($\mathcal{NP}$-complet).

Il est donc difficile (si $\mathcal{P} \neq \mathcal{NP}$) de décider rapidement si ϕ est satisfaisable ou pas quand $k \geq 3$.

Le problème k-SAT

Petit rappel:

Problème (k-SAT)

Etant donné ϕ, une formule booléenne k-CFN, décider s'il existe une assignation qui satisfait ϕ.

- Pour $k = 2$, le problème est facile (dans $\mathcal{P}$).
- Pour $k \geq 3$, moins facile ($\mathcal{NP}$-complet).

Il est donc difficile (si $\mathcal{P} \neq \mathcal{NP}$) de décider rapidement si ϕ est satisfaisable ou pas quand $k \geq 3$.

$\Rightarrow$ Le lemme local de Lovász nous permet de reconnaître rapidement certaines k-CNF satisfaisables, juste en regardant leurs structures.

Le problème k-SAT

Petit rappel:

Problème (k-SAT)

Etant donné ϕ, une formule booléenne k-CFN, décider s'il existe une assignation qui satisfait ϕ.

- Pour $k = 2$, le problème est facile (dans $\mathcal{P}$).
- Pour $k \geq 3$, moins facile ($\mathcal{NP}$-complet).

Il est donc difficile (si $\mathcal{P} \neq \mathcal{NP}$) de décider rapidement si ϕ est satisfaisable ou pas quand $k \geq 3$.

$\Rightarrow$ Le lemme local de Lovász nous permet de reconnaître rapidement certaines k-CNF satisfaisables, juste en regardant leurs structures.

Dans la suite nous dirons que deux clauses s'intersectent si elles partagent au moins une variable.

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

Démonstration.

Prenons une assignation aléatoire des variables booléennes.

- A_i représente l'évènement: "la i-ème clause est non satisfaite".

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

Démonstration.

Prenons une assignation aléatoire des variables booléennes.

- A_i représente l'évènement: "la i-ème clause est non satisfaite".
- $\Pr[A_i] = 2^{-k} = p$.

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

Démonstration.

Prenons une assignation aléatoire des variables booléennes.

- A_i représente l'évènement: "la i-ème clause est non satisfaite".
- $\Pr[A_i] = 2^{-k} = p$.
- Chaque A_i est mutuellement indépendant avec l'ensemble de tous les A_j tel que la i-ème et la j-ème clause ne s'intersectent pas ($d \leq 2^{k-2}$).

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

Démonstration.

Prenons une assignation aléatoire des variables booléennes.

- A_i représente l'évènement: "la i-ème clause est non satisfaite".
- $\Pr[A_i] = 2^{-k} = p$.
- Chaque A_i est mutuellement indépendant avec l'ensemble de tous les A_j tel que la i-ème et la j-ème clause ne s'intersectent pas ($d \leq 2^{k-2}$).

Comme $4pd \leq 42^{k-2}2^k = 1$, le lemme local nous dit que ϕ sera satisfaite avec une probabilité non nulle. □

LLL et k-SAT

Proposition

Soit ϕ une formule k-CNF. Si chaque clause intersecte au plus 2^{k-2} autres clauses, alors ϕ est satisfaisable.

Démonstration.

Prenons une assignation aléatoire des variables booléennes.

- A_i représente l'évènement: "la i-ème clause est non satisfaite".
- $\Pr[A_i] = 2^{-k} = p$.
- Chaque A_i est mutuellement indépendant avec l'ensemble de tous les A_j tel que la i-ème et la j-ème clause ne s'intersectent pas ($d \leq 2^{k-2}$).

Comme $4pd \leq 42^{k-2}2^k = 1$, le lemme local nous dit que ϕ sera satisfaite avec une probabilité non nulle. □

$\Rightarrow$ Preuve non constructive.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

(1991) Alon : formule avec au plus $2^{\frac{k}{8}}$ intersections. Simplifie l'algorithme de Beck en introduisant de l'aléatoire dans celui-ci.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

(1991) Alon : formule avec au plus $2^{\frac{k}{8}}$ intersections. Simplifie l'algorithme de Beck en introduisant de l'aléatoire dans celui-ci.

(2000) Czumaj et Scheideler : présentent une variante de la méthode de Alon qui fonctionne pour le cas non-uniforme.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

(1991) Alon : formule avec au plus $2^{\frac{k}{8}}$ intersections. Simplifie l'algorithme de Beck en introduisant de l'aléatoire dans celui-ci.

(2000) Czumaj et Scheideler : présentent une variante de la méthode de Alon qui fonctionne pour le cas non-uniforme.

(2008) Srinivasan : formule avec au plus $2^{\frac{k}{4}}$ intersections.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

(1991) Alon : formule avec au plus $2^{\frac{k}{8}}$ intersections. Simplifie l'algorithme de Beck en introduisant de l'aléatoire dans celui-ci.

(2000) Czumaj et Scheideler : présentent une variante de la méthode de Alon qui fonctionne pour le cas non-uniforme.

(2008) Srinivasan : formule avec au plus $2^{\frac{k}{4}}$ intersections.

(2009) Moser : formule avec au plus 2^{k-5} intersections.

Ligne du temps

La preuve précédente ne donne aucun indice sur comment trouver une assignation satisfaisante en un temps raisonnable. Il y a eu plusieurs tentatives.

(1991) Beck : formule avec au plus $2^{\frac{k}{48}}$ intersections.

(1991) Alon : formule avec au plus $2^{\frac{k}{8}}$ intersections. Simplifie l'algorithme de Beck en introduisant de l'aléatoire dans celui-ci.

(2000) Czumaj et Scheideler : présentent une variante de la méthode de Alon qui fonctionne pour le cas non-uniforme.

(2008) Srinivasan : formule avec au plus $2^{\frac{k}{4}}$ intersections.

(2009) Moser : formule avec au plus 2^{k-5} intersections.

(2010) Moser et Tardos : donnent un algorithme randomisé pour la version générale du lemme local de Lovász.

Exemple introductif

Considérons les strings suivants:

String 1: 010101010101010101010101
String 2: 110100110010110100101100
String 3: 100111011101011100100110

Ils ont tous 24 bits, mais il existe une différence dans leur "description" respective.

Exemple introductif

Considérons les strings suivants:

String 1: 010101010101010101010101
String 2: 110100110010110100101100
String 3: 100111011101011100100110

Ils ont tous 24 bits, mais il existe une différence dans leur "description" respective.

- Le string 1 peut être décrit comme étant un string de 12 blocs de "01".

Exemple introductif

Considérons les strings suivants:

String 1: 010101010101010101010101

String 2: 110100110010110100101100

String 3: 100111011101011100100110

Ils ont tous 24 bits, mais il existe une différence dans leur "description" respective.

- Le string 1 peut être décrit comme étant un string de 12 blocs de "01".
- Le string 2 a la propriété d'avoir un "1" en i-ème position si et seulement si l'écriture de i en binaire contient un nombre impair de 1.

Exemple introductif

Considérons les strings suivants:

String 1: 010101010101010101010101

String 2: 110100110010110100101100

String 3: 100111011101011100100110

Ils ont tous 24 bits, mais il existe une différence dans leur "description" respective.

- Le string 1 peut être décrit comme étant un string de 12 blocs de "01".
- Le string 2 a la propriété d'avoir un "1" en i-ème position si et seulement si l'écriture de i en binaire contient un nombre impair de 1.
- Le string 3 ne semble pas avoir de description similaire, pour le décrire complètement il faut donc le donner entièrement.

Définition

$\Rightarrow$ Qu'est-ce qu'une description ?

Définition

$\Rightarrow$ Qu'est-ce qu'une description ?

Fixons $\Sigma = \{0, 1\}$.

Définition

Soit $f : \Sigma^* \to \Sigma^*$, alors une description (relative à f) d'un string σ est un string τ tel que $f(\tau) = \sigma$. Nous demandons également que f soit calculable.

Définition

$\Rightarrow$ Qu'est-ce qu'une description ?

Fixons $\Sigma = \{0, 1\}$.

Définition

Soit $f : \Sigma^* \to \Sigma^*$, alors une description (relative à f) d'un string σ est un string τ tel que $f(\tau) = \sigma$. Nous demandons également que f soit calculable.

Ce qui nous donne la définition de la complexité de Kolmogorov

$$K_f(x) = \begin{cases} \min\{|p| : f(p) = x\} & \text{si } x \in \operatorname{Img} f \\ \infty & \text{sinon} \end{cases}$$

Définition

$\Rightarrow$ Qu'est-ce qu'une description ?

Fixons $\Sigma = \{0, 1\}$.

Définition

Soit $f : \Sigma^* \to \Sigma^*$, alors une description (relative à f) d'un string σ est un string τ tel que $f(\tau) = \sigma$. Nous demandons également que f soit calculable.

Ce qui nous donne la définition de la complexité de Kolmogorov

$$K_f(x) = \begin{cases} \min\{|p| : f(p) = x\} & \text{si } x \in \operatorname{Img} f \\ \infty & \text{sinon} \end{cases}$$

Notons que cette définition de la complexité dépend de f, ce problème peut être contourné à l'aide d'une Machine de Turing Universelle.

Kolmogorov aléatoire

Théorème

Pour tout n, il existe un x avec $|x| = n$ tel que $K(x) \geq n$.

De tels x sont appelés Kolmogorov aléatoires.

Kolmogorov aléatoire

Théorème

Pour tout n, il existe un x avec $|x| = n$ tel que $K(x) \geq n$.

De tels x sont appelés Kolmogorov aléatoires.

Démonstration.

Par l'absurde.

- Pour tout x, $K(x) < n$ et donc il existe p_x tel que $g(p_x) = x$ et $|p_x| < n$. Notons que si $x \neq y$ alors $p_x \neq p_y$.

Kolmogorov aléatoire

Théorème

Pour tout n, il existe un x avec $|x| = n$ tel que $K(x) \geq n$.

De tels x sont appelés Kolmogorov aléatoires.

Démonstration.

Par l'absurde.

- Pour tout x, $K(x) < n$ et donc il existe p_x tel que $g(p_x) = x$ et $|p_x| < n$. Notons que si $x \neq y$ alors $p_x \neq p_y$.
- Il y a $2^n - 1$ programmes de taille inférieure à n et 2^n strings de taille n.

Kolmogorov aléatoire

Théorème

Pour tout n, il existe un x avec $|x| = n$ tel que $K(x) \geq n$.

De tels x sont appelés Kolmogorov aléatoires.

Démonstration.

Par l'absurde.

- Pour tout x, $K(x) < n$ et donc il existe p_x tel que $g(p_x) = x$ et $|p_x| < n$. Notons que si $x \neq y$ alors $p_x \neq p_y$.
- Il y a $2^n - 1$ programmes de taille inférieure à n et 2^n strings de taille n.

En utilisant le principe des tiroirs de Dirichlet, nous obtenons une contradiction. □

Kolmogorov aléatoire

Théorème

Pour tout n, il existe un x avec $|x| = n$ tel que $K(x) \geq n$.

De tels x sont appelés Kolmogorov aléatoires.

Démonstration.

Par l'absurde.

- Pour tout x, $K(x) < n$ et donc il existe p_x tel que $g(p_x) = x$ et $|p_x| < n$. Notons que si $x \neq y$ alors $p_x \neq p_y$.
- Il y a $2^n - 1$ programmes de taille inférieure à n et 2^n strings de taille n.

En utilisant le principe des tiroirs de Dirichlet, nous obtenons une contradiction. □

Un string x est compressible par un nombre c s'il possède une description de taille inférieure à $|x| - c$.

Kolmogorov aléatoire (2)

Remarque La "plupart" des strings ne sont pas significativement compressible.

Kolmogorov aléatoire (2)

Remarque La "plupart" des strings ne sont pas significativement compressible.

Précisions cette remarque,

- pour n fixé, il y a 2^n strings binaires de taille n,

Kolmogorov aléatoire (2)

Remarque La "plupart" des strings ne sont pas significativement compressible.

Précisions cette remarque,

- pour n fixé, il y a 2^n strings binaires de taille n,
- choisissons un string x de façon uniformément aléatoire avec probabilité 2^{-n},

Kolmogorov aléatoire (2)

Remarque La "plupart" des strings ne sont pas significativement compressible.

Précisions cette remarque,

- pour n fixé, il y a 2^n strings binaires de taille n,
- choisissons un string x de façon uniformément aléatoire avec probabilité 2^{-n},
- la probabilité que x soit compressible par c est négligeable ; elle vaut $2^{-c+1} - 2^{-n}$.

Kolmogorov aléatoire (2)

Remarque La "plupart" des strings ne sont pas significativement compressible.

Précisions cette remarque,

- pour n fixé, il y a 2^n strings binaires de taille n,
- choisissons un string x de façon uniformément aléatoire avec probabilité 2^{-n},
- la probabilité que x soit compressible par c est négligeable ; elle vaut $2^{-c+1} - 2^{-n}$.

Cette dernière valeur vient du fait que le nombre de strings dont la taille est inférieure à $n - c$ est donnée par la série géométrique

$$1 + 2 + 2^2 + \ldots + 2^{n-c} = 2^{n-c+1} - 1.$$

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

- prouver qu'il existe une infinité de nombres premiers,

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

- prouver qu'il existe une infinité de nombres premiers,
- preuve du théorème d'incomplétude de Gödel,

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

- prouver qu'il existe une infinité de nombres premiers,
- preuve du théorème d'incomplétude de Gödel,
- obtenir une borne inférieur pour le temps nécessaire dans la reconnaissance des palindromes,

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

- prouver qu'il existe une infinité de nombres premiers,
- preuve du théorème d'incomplétude de Gödel,
- obtenir une borne inférieur pour le temps nécessaire dans la reconnaissance des palindromes,
- lien avec l'entropie de Shannon,

Application de la complexité de Kolmogorov

Quelques applications plus ou moins immédiates:

- prouver qu'il existe une infinité de nombres premiers,
- preuve du théorème d'incomplétude de Gödel,
- obtenir une borne inférieur pour le temps nécessaire dans la reconnaissance des palindromes,
- lien avec l'entropie de Shannon,
- thermodynamique.

La minute xkcd

Enoncé

Théorème

Il existe une constante c telle que, étant donné ϕ une formule k-CNF de m clauses, avec la condition qu'aucune d'entre elles ne possèdent plus de $r = 2^{k-c}$ intersections, il est possible de trouver une assignation qui satisfait ϕ en temps polynomial (en m).

Enoncé

Théorème

Il existe une constante c telle que, étant donné ϕ une formule k-CNF de m clauses, avec la condition qu'aucune d'entre elles ne possèdent plus de $r = 2^{k-c}$ intersections, il est possible de trouver une assignation qui satisfait ϕ en temps polynomial (en m).

Moser utilise l'algorithme randomisé `Solve`(ϕ) constitué d'appels à une procédure récursive `Fix`(C) pour les clauses C de ϕ.

Enoncé

Théorème

Il existe une constante c telle que, étant donné ϕ une formule k-CNF de m clauses, avec la condition qu'aucune d'entre elles ne possèdent plus de $r = 2^{k-c}$ intersections, il est possible de trouver une assignation qui satisfait ϕ en temps polynomial (en m).

Moser utilise l'algorithme randomisé `Solve`(ϕ) constitué d'appels à une procédure récursive `Fix`(C) pour les clauses C de ϕ.

$\Rightarrow$ Nous allons montrer ici une idée intuitive du raisonnement de Moser.

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

La fonction `Fix`(C)

- Remplacer les valeurs des variables dans C par de nouvelles valeurs aléatoires ;

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

La fonction `Fix`(C)

- Remplacer les valeurs des variables dans C par de nouvelles valeurs aléatoires ;
- Tant qu'il existe une clause D non satisfaite qui partage des variables avec C, appeler `Fix`(D).

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

La fonction `Fix`(C)

- Remplacer les valeurs des variables dans C par de nouvelles valeurs aléatoires ;
- Tant qu'il existe une clause D non satisfaite qui partage des variables avec C, appeler `Fix`(D).

Quelques remarques:

- Si `Fix`(C) termine, toute clause satisfaite avant l'appel de `Fix`(C) est encore satisfaite.

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

La fonction `Fix`(C)

- Remplacer les valeurs des variables dans C par de nouvelles valeurs aléatoires ;
- Tant qu'il existe une clause D non satisfaite qui partage des variables avec C, appeler `Fix`(D).

Quelques remarques:

- Si `Fix`(C) termine, toute clause satisfaite avant l'appel de `Fix`(C) est encore satisfaite.
- `Solve`(ϕ) fait au plus m appel à `Fix`.

Algorithme

La fonction `Solve`(ϕ)

- Choisir une assignation aléatoire $a \in \{0,1\}^n$;
- Tant qu'il existe des clauses C non satisfaites, appeler `Fix`(C).

La fonction `Fix`(C)

- Remplacer les valeurs des variables dans C par de nouvelles valeurs aléatoires ;
- Tant qu'il existe une clause D non satisfaite qui partage des variables avec C, appeler `Fix`(D).

Quelques remarques:

- Si `Fix`(C) termine, toute clause satisfaite avant l'appel de `Fix`(C) est encore satisfaite.
- `Solve`(ϕ) fait au plus m appel à `Fix`.

Nous voulons montrer que tous les `Fix`(C) se terminent.

$\Rightarrow$ Supposons que l'algorithme appel `Fix`(C) au moins s fois (avec les appels récursifs). Moser montre que s est borné par $O(m \log m)$.

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

$$x = \underbrace{1011\ldots01}_{n} + \overbrace{\underbrace{1110\ldots00}_{k} + \underbrace{1000\ldots01}_{k} + \ldots + \underbrace{0110\ldots11}_{k}}^{sk}$$

$$\ldots \wedge (x_1 \vee \ldots \vee x_k) \wedge (x_{k_1} \vee \ldots \vee x_{k_2}) \wedge (x_{k_3} \vee \ldots \vee x_{k_4}) \wedge \ldots$$

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

$$x = \underbrace{1011\ldots01}_{n} + \overbrace{\underbrace{1110\ldots00}_{k} + \underbrace{1000\ldots01}_{k} + \ldots + \underbrace{0110\ldots11}_{k}}^{sk}$$

$$\ldots \wedge (a_1 \vee \ldots \vee a_k) \wedge (a_{k_1} \vee \ldots \vee a_{k_2}) \wedge (a_{k_3} \vee \ldots \vee a_{k_4}) \wedge \ldots$$

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

$$x = \underbrace{1011\ldots01}_{n} + \overbrace{\underbrace{1110\ldots00}_{k} + \underbrace{1000\ldots01}_{k} + \ldots + \underbrace{0110\ldots11}_{k}}^{sk}$$

$$\ldots \wedge (a'_1 \vee \ldots \vee a'_k) \wedge (a_{k_1} \vee \ldots \vee a_{k_2}) \wedge (a_{k_3} \vee \ldots \vee a_{k_4}) \wedge \ldots$$

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

$$x = \underbrace{1011\ldots01}_{n} + \overbrace{\underbrace{1110\ldots00}_{k} + \underbrace{1000\ldots01}_{k} + \ldots + \underbrace{0110\ldots11}_{k}}^{sk}$$

$$\ldots \wedge (a'_1 \vee \ldots \vee a'_k) \wedge (a'_{k_1} \vee \ldots \vee a'_{k_2}) \wedge (a_{k_3} \vee \ldots \vee a_{k_4}) \wedge \ldots$$

Argument d'incompressibilité

Soit un x string aléatoire de taille $n + sk$ (n est le nombre total de variables).
Supposons que l'algorithme utilise

- les premiers n bits de x pour l'assignation initiale.
- k bits pour remplacer les valeurs des variables dans chaque appel de `Fix`.

$$x = \underbrace{1011\ldots01}_{n} + \overbrace{\underbrace{1110\ldots00}_{k} + \underbrace{1000\ldots01}_{k} + \ldots + \underbrace{0110\ldots11}_{k}}^{sk}$$

$$\ldots \wedge (a'_1 \vee \ldots \vee a'_k) \wedge (a'_{k_1} \vee \ldots \vee a'_{k_2}) \wedge (a_{k_3} \vee \ldots \vee a_{k_4}) \wedge \ldots$$

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

- Les clauses C tels que `Fix`(C) est appelé par `Solve` peuvent être décrites avec $O(m \log m)$ bits.

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

- Les clauses C tels que `Fix`(C) est appelé par `Solve` peuvent être décrites avec $O(m \log m)$ bits.
- Chacune des autres clauses peut être décrite en $\log r + \alpha$ (pour une constante α).

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

- Les clauses C tels que `Fix(`C`)` est appelé par `Solve` peuvent être décrites avec $O(m \log m)$ bits.
- Chacune des autres clauses peut être décrite en $\log r + \alpha$ (pour une constante α).

Comme x était Kolomogorov aléatoire, nous avons que

$$O(m \log m) + s(\log r + \alpha) + n \geq n + sk,$$

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

- Les clauses C tels que `Fix(`C`)` est appelé par `Solve` peuvent être décrites avec $O(m \log m)$ bits.
- Chacune des autres clauses peut être décrite en $\log r + \alpha$ (pour une constante α).

Comme x était Kolomogorov aléatoire, nous avons que

$$O(m \log m) + s(\log r + \alpha) + n \geq n + sk,$$
$$s(k - \log r - \alpha) \leq O(m \log m).$$

Argument d'incompressibilité (2)

Remarquons que si nous savons quelles clauses sont réparées par `Fix`, alors nous connaissons l'assignation des ses clauses (avant l'appel).

$\Rightarrow$ x peut être décrit par la liste des clauses réparées et n bits de l'assignation finale.

- Les clauses C tels que `Fix(`C`)` est appelé par `Solve` peuvent être décrites avec $O(m \log m)$ bits.
- Chacune des autres clauses peut être décrite en $\log r + \alpha$ (pour une constante α).

Comme x était Kolomogorov aléatoire, nous avons que

$$O(m \log m) + s(\log r + \alpha) + n \geq n + sk,$$
$$s(k - \log r - \alpha) \leq O(m \log m).$$

Pour montrer que s est borné, il faut que $(k - \log r - \alpha) \geq 0$, donc que $r \leq 2^{k-c}$ pour $c > \alpha$.

Merci de votre écoute

Théorème (généralisé)

Considérons un ensemble $\mathcal{E} = \{A_1, A_2, \ldots, A_n\}$ *d'événements tel que chaque* A_i *est mutuellement indépendant de chaque évènement dans* $\mathcal{E} \backslash (D_i \cup A_i)$, *pour* $D_i \subseteq \mathcal{E}$. *Soit des réels* $x_1, \ldots, x_n \in [0, 1[$ *tels que pour chaque* $1 \leq i \leq n$:

$$\Pr[A_i] \leq x_i \prod_{A_j \in D_j} (1 - x_j),$$

alors la probabilité qu'aucun des événements de $\mathcal{E}$ *ne se produise est au moins* $\prod_{i=1}^{n}(1 - x_i) > 0$

Théorème (asymétrique)

Considérons un ensemble $\mathcal{E} = \{A_1, A_2, \ldots, A_n\}$ d'événements tel que chaque A_i est mutuellement indépendant de chaque évènement dans $\mathcal{E} \setminus (D_i \cup A_i)$, pour $D_i \subseteq \mathcal{E}$. Pour chaque $1 \leq i \leq n$

- $\Pr[A_i] \leq \frac{1}{4}$,
- $\sum_{A_j \in D_j} \Pr[A_j] \leq \frac{1}{4}$.

alors avec une probabilité positive, aucun des événements de $\mathcal{E}$ ne se produit.

Théorème (pondéré)

Considérons un ensemble $\mathcal{E} = \{A_1, A_2, \ldots, A_n\}$ d'événements tel que chaque A_i est mutuellement indépendant de chaque évènement dans $\mathcal{E} \setminus (D_i \cup A_i)$, pour $D_i \subseteq \mathcal{E}$. Soit des entiers $t_1, \ldots, t_n \geq 0$ et un réel $0 \leq p \leq \frac{1}{4}$ tels que, pour chaque $1 \leq i \leq n$:

- $\Pr[A_i] \leq p^{t_i}$,
- $\sum_{A_j \in D_j} (2p)^{t_j} \leq \frac{t_i}{2}$.

alors avec une probabilité positive, aucun des événements de $\mathcal{E}$ ne se produit.

Théorème (inégal)

Considérons un ensemble $\mathcal{E}$ d'événements tel que, pour chaque $A \in \mathcal{E}$, il existe un ensemble $D(A)$ d'au plus d autres événements tel que pour tout sous-ensemble $S \subseteq \mathcal{E} \backslash (A \cup D(A))$, on a

$$\Pr[\bigcap_{A_j \in S} A_j^{\mathbf{C}}] \leq p.$$

Si $4pd \leq 1$, alors avec une probabilité positive, aucun des événements de $\mathcal{E}$ ne se produit.

Théorème

Il existe une infinité de nombres premiers.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

- Comme $e_i \leq \log m$, alors $|e_i| \leq \log \log m$.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

- Comme $e_i \leq \log m$, alors $|e_i| \leq \log \log m$.
- $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log \log m$.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

- Comme $e_i \leq \log m$, alors $|e_i| \leq \log\log m$.
- $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log\log m$.
- Comme $m \leq 2^{n+1}$, alors $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log(n+1)$.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

- Comme $e_i \leq \log m$, alors $|e_i| \leq \log\log m$.
- $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log\log m$.
- Comme $m \leq 2^{n+1}$, alors $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log(n+1)$.
- Donc, $K(m) \leq 2k \log(n+1) + c$.

Théorème

Il existe une infinité de nombres premiers.

Démonstration.

Par l'absurde, il existe $p_1, \ldots, p_k$ tels que pour tout m:

$$m = p_1^{e_1} \ldots p_k^{e_k}$$

les e_i décrivent m. Soit m aléatoire de taille n.

- Comme $e_i \leq \log m$, alors $|e_i| \leq \log\log m$.
- $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log\log m$.
- Comme $m \leq 2^{n+1}$, alors $|e_1 \oplus \ldots \oplus e_k| \leq 2k \log(n+1)$.
- Donc, $K(m) \leq 2k \log(n+1) + c$.

Contradiction avec le fait que m soit aléatoire. □

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.
- Soit $S_c(x)$ la formule "x est le plus petit string de taille c avec $K(x) \geq c$".

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.
- Soit $S_c(x)$ la formule "x est le plus petit string de taille c avec $K(x) \geq c$".
- Il en suit que $K(S_c(x)) \leq \alpha \log c$, pour une constante α (pas de dépendance avec c ou T).

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.
- Soit $S_c(x)$ la formule "x est le plus petit string de taille c avec $K(x) \geq c$".
- Il en suit que $K(S_c(x)) \leq \alpha \log c$, pour une constante α (pas de dépendance avec c ou T).
- Il est possible de montre qu'il existe un unique x tel que $S_c(x) = \mathit{true}$ est vrai.

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.
- Soit $S_c(x)$ la formule "x est le plus petit string de taille c avec $K(x) \geq c$".
- Il en suit que $K(S_c(x)) \leq \alpha \log c$, pour une constante α (pas de dépendance avec c ou T).
- Il est possible de montre qu'il existe un unique x tel que $S_c(x) = \mathit{true}$ est vrai.
- Nous obtenons une description de x avec la description de T et de $S_c(x)$.

Théorème

Tout système formel consistant qui inclus de l'arithmétique contient des théorèmes qui ne peuvent pas être prouvés dans le système lui-même.

Démonstration.

- Soit T une théorie axiomatique consistante qui contient de l'arithmétique descriptible en k bits.
- Supposons que T est complet.
- Soit $S_c(x)$ la formule "x est le plus petit string de taille c avec $K(x) \geq c$".
- Il en suit que $K(S_c(x)) \leq \alpha \log c$, pour une constante α (pas de dépendance avec c ou T).
- Il est possible de montre qu'il existe un unique x tel que $S_c(x) = \mathit{true}$ est vrai.
- Nous obtenons une description de x avec la description de T et de $S_c(x)$.

Il est possible de montrer que dans ce cas, $K(x) \leq 2k + \log c + d$ (pour une constante d), qui contredit le fait que $K(x) \geq c$. □