Théorème du Speed-Up de Blum

Tom Sarry

Université de Montréal

30 Novembre 2022

Table des Matières

Table des Matières

Contexte

- Turing 1936 – Machines de Turing
- Blum 1967 – Théorème du Speed-Up
- Cook 1971 – Comment prouver que $X \in \mathrm{NP}$
- Young 1973 – Preuve simplifiée du Speed-Up
- Cutland 1980 – Manuel de cours pour la théorie des fonctions recursives, reprennant la preuve de Young pour le théorème du Speed-Up.

Manuel Blum

Figure: Manuel Blum en 2018, © Jan Gott

- BSc, MSc, PhD (Dr. Minsky) au MIT
- 1995 Prix Turing – cryptographie, théorèmes du Speed-Up / Compression, (re)CAPTCHA, théorie de complexité de calcul...
- Professeur à Berkeley, Carnegie Mellon

Énoncé du théorème

Speed-Up

Il existe des fonctions pour lesquelles aucun programme les calculant n'est *optimal*.

- Il existe une fonction telle que, si l'on me donne un programme la calculant, il existera toujours un programme 2 fois plus rapide.
- ..., il existera toujours un programme utilisant 10 fois moins d'espace.
- ...

Intuition

Calcul de $f : w \mapsto w$ est un palindrome (par une MT).

- $f(\text{racecar}) = 1$
- $f(\text{blum}) = 0$
- $f(a^{10^{10}}) = 1$

Algorithme:

`vérifier w[0]=w[len(w)-1], vérifier w[1]=w[len(w-2)] ...`

Meilleur Algorithme (pour longs mots):

`Sauvegarder w[0], w[1], aller à la fin du mot et vérifier les deux dernières cases...`

Encore Meilleur Algorithme (pour encore plus longs mots):

`Sauvegarder w[0], w[1], w[2], w[3], aller à la fin du mot et vérifier les 4 dernières cases...`

Table des Matières

Terminologie

- $\{P_i\}$: Énumération de programmes (MT binaire ou base 10, python ou C...).
- $\phi_a^{(n)}$ est la fonction n-aire calculée par P_a.
- $\{\phi_i\}$: Énumération de fonctions calculées par des programmes.
- Pour chaque fonction unaire calculable f, $\exists$ un programme qui calcule f, donc $f = \phi_a$. a est un **indice pour** f.

Terminologie

- Extensionalité: $f \simeq g$ si leur fonctionnement observé est le même, eg
  - $f(n) = (n+5) \times 2$
  - $g(n) = (n \times 2) + 10$
- $\{\Phi_i\}$ est une **mesure de complexité de calcul** si:
  (a) $\mathrm{Dom}(\Phi_e) = \mathrm{Dom}(\phi_e)$, $\forall e$.
  (b) Le prédicat $\Phi_e(x) \simeq y$ est décidable.
- $P_i = \langle \phi_i, \Phi_i \rangle$, eg. $P_0(5) = 1$, en 7 étapes.
- $t_e^{(n)} = \mu t[P_e(x) \downarrow$ en t étapes$]$, est une mesure de complexité de calcul.
- *p.p.*: presque partout = faux pour uniquement un nombre fini d'éléments.

Table des Matières

Pseudo Théorème du Speed-Up

Soit r une fonction totale calculable. Il existe une fonction totale calculable f t.q. pour n'importe quel programme P_i pour f, nous pouvons trouver un P_j satisfaisant:

(a) ϕ_j est totale et $\phi_j(x) = f(x)$ *p.p.*

(b) $r(t_j(x)) < t_i(x)$ *p.p.*(généralisable pour Φ_j)

Pseudo Théorème du Speed-Up

Théorème s-m-n: [Cutland 4.3]

Pour tout $m, n \geq 1$, il existe une $m+1$-aire fonction totale calculable $s_n^m(e, x)$, satisfaisant:

$$\phi_e^{(m+n)}(x, y) \simeq \phi_{s_n^m(e,x)}^{(n)}(y)$$

- Fixons s donné par s-m-n, t.q. $\phi_e^{(2)}(u, x) \simeq \phi_{s(e,u)}(x)$.
- But: soit $g_u(x) = \phi_e^{(2)}(u, x)$, trouver l'indice e t.q. $\phi_e^{(2)}$ est totale, satisfaisant:
  (a) $g_0 = f$
  (b) $\forall u,\ g_u(x) = g_0(x)$ *p.p.*
  (c) si $f = \phi_i$, alors il existe un indice j pour g_{i+1} t.q. $r(t_j(x)) < t_i(x)$ *p.p.*
- Note: Conditions suffisantes pour prouver le théorème

Pseudo Théorème du Speed-Up

Pour obtenir g, nous définissons un groupe d'ensembles finis d'*indices annulés* $C_{u,0}, C_{u,1}, ..., C_{u,x-1}$.

$$C_{u,x} = \begin{cases} \{i : u \leq i < x, i \notin \bigcup_{y<x} C_{u,y} \text{ et } t_i(x) \leq r(t_{s(e,i+1)}(x))\} & \\ & \text{si } t_{s(e,i+1)}(x) \text{ est défini pour } u \leq i < x \\ \text{undefined} & \text{sinon.} \end{cases}$$

Ainsi nous pouvons obtenir $g(u,x)$:

$$g(u,x) = \begin{cases} 1 + \max\{\phi_i(x) : i \in C_{u,x}\} & \text{si } C_{u,x} \text{ est défini} \\ \text{undefined} & \text{sinon.} \end{cases}$$

$g_u(x)$ obtenue par diagonalisation, différente de toutes les fonctions annulées dans $\bigcup_x C_{u,x}$.

Exemple

Supposons:

- $C_{0,5} = \{2, 4\}$
- $\phi_2(5) = 12$
- $\phi_4(5) = 6$

$$g_0(5) = 1 + \mathsf{max}\{\phi_i(5) : i \in \{2, 4\}\}$$
$$g_0(5) = 13$$
$$g_0(x) \neq \phi_2(x) \wedge g_0(x) \neq \phi_4(x)$$

Pseudo Théorème du Speed-Up

Corollaire du 2^e théorème de la récursion [Cutland 11.1.4]

Soit $h(x, y)$ une fonction calculable arbitraire, alors il existe un indice e t.q.

$$h(e, y) \simeq \phi_e(y)$$

- g dépend implicitement de e, donc $g(u, x) = h(e, u, x)$.
- $h(e, u, x) \simeq \phi_e^{(2)}(u, x)$ en généralisant le corollaire ci-dessus. $(\star)$
- Fixons e car c'est l'indice requis pour la preuve, vérifions ses propriétés.

$g(u,x)$ totale

- Pour $u \geq x, C_{u,x} = \emptyset \implies g(u,x) = 1$.
- Pour $u < x$, preuve par récurrence forte inverse (fixer x):
  1. **Initialisation –** $g(x,x)$ définie par le point ci-dessus.
  2. **Hérédité –** Supposons que $g(x,x), g(x-1,x), \cdots, g(u+1,x)$ définies.
     $\implies \phi_e(x,x), \cdots, \phi_e(u+1,x)$ définies (cf. $\star$)
     $\implies \phi_{s(e,x)}(x), \cdots, \phi_{s(e,u+1)}(x)$ définies (cf. s)
     $\implies t_{s(e,i+1)}(x), u \leq i < x$ définies
     $\implies C_{u,x}$ définie
     $\implies g(u,x)$ définie
  3. **Conclusion –** $g(u,x)$ est totale. $\square$

(a) $g_0 = f$

Définissons $g_u(x) = g(u, x)$. Alors, $g_u(x) = \phi_{s(e,u)}(x)$ par $(\star, s)$.
Assignons $f = g_0$, f est donc totale.

(b) $\forall u,\ g_u(x) = g_0(x)$ *p.p.*

Fixons un u et montrons que $g(0,x) = g(u,x)$ *p.p.*

- $C_{u,x} = C_{0,x} \cap \{u, u+1, \cdots, x-1\}$ (visible par construction)
- $C_{0,x}$ tous disjoints par construction, on peut trouver $v = \max\{x : C_{0,x}$ contient un indice $i < u\}$.
- Pour $x > v$, $C_{0,x} \subseteq \{u, u+1, \cdots, x-1\}$, et donc $C_{0,x} = C_{u,x}$.
- $\implies g(0,x) = g(u,x)$ for $x > v$.
- $\implies g_0(x) = g_u(x)$ *p.p.*

(c) si $f = \phi_i, \exists$ un indice j pour g_{i+1} t.q. $r(t_j(x)) < t_i(x)$ *p.p.*

Soit i un indice pour f, prenons $j = s(e, i+1)$.

1. $\phi_j = \phi_{s(e,i+1)} = g_{i+1}$, donc j est un indice pour g_{i+1}.
2. $r(t_j(x)) = r(t_{s(e,i+1)}(x)) < t_i(x),\ \forall x > i$. Sinon, i aurait été annulé dans la définition de $g(0, x)$ pour un $x > i$.

$$g(0,x) = 1 + \max\{\phi_k(x) : k \in C_{0,x}\}$$
$$g(0,x) \neq \phi_i(x) \Rightarrow\Leftarrow$$

Les propriétés sont vraies pour cet indice e. □

Table des Matières

Soit r une fonction totale calculable (croissante monotone). Il existe une fonction totale calculable f t.q. pour n'importe quel programme P_i pour f, nous pouvons trouver un P_k satisfaisant:

(a) ϕ_k est totale et $\phi_k(x) = f(x)$

(b) $r(t_k(x)) < t_i(x)$ *p.p.*(généralisable pour Φ_k)

Théorème du Speed-Up

- Selon le Pseudo Théorème du Speed-Up, nous trouvons un P_j satisfaisant:
  (a) ϕ_j est totale et $\phi_j(x) = f(x)$ *p.p.*
  (b) $r(t_j(x)) < t_i(x)$ *p.p.*
- Il existe c à partir duquel $\phi_j(x) = f(x), \forall x > c$.
- Créer un P_{j^*} avec $f(0), ..., f(c)$ dans sa boîte de contrôle, et $\phi_j(x)$, pour $x > c$.
- Prenons $k = j^*$, car $\phi_{j^*} = f$.
- Vu que nous avons ajouté un nombre constant d'instructions à P_{j^*} pour un nombre fini d'éléments, nous avons:
  - $t_{j^*}(x) \leq t_j(x) + c$
  - $r(t_{j^*}(x)) \leq r(t_j(x) + x) < t_i(x)$ *p.p.* $\square$

Table des Matières

Conclusion

- Le Pseudo Théorème du Speed-Up est *efficace*: avec P qui calcule f, on peut trouver ce *meilleur* programme qui calcule f *p.p.*, et est plus rapide que P *p.p.*
- Ce n'est pas le cas pour le Théorème du Speed-Up.
- **Théorème de la Compression** – réciproque du Speed-Up
  Pour n'importe quel ϕ_i, il existe une fonction f définie sur le même domaine étant si complexe que n'importe quelle machine calculant $f(n)$ prendra au moins $\Phi_i(n)$ étapes *p.p.*, mais il existe au moins un programme qui calcule $f(n)$ en moins de $[\Phi_i(n)]^7$ étapes.

Sources

- Blum M. [67], *A Machine-Independent Theory of the Complexity of Recursive Functions*
- Young P. [73], *Easy Construction in Complexity Theory: Gap and Speed-Up Theorems*
- Cutland N. [80], *COMPUTABILITY An introduction to recursive function theory*