Réductions *non-boîtes noires* & différentiation automatique

Pierre Karpman

Lycée Champollion TD XENS

https://membres-ljk.imag.fr/Pierre.Karpman/CPGE/2025/

Table des matières

Réductions

Réduire $\mathcal{B}$ à $\mathcal{A}$: « si l'on sait résoudre $\mathcal{A}$, alors on peut résoudre $\mathcal{B}$ »

(Pseudo-)formalisation avec oracle

Une *réduction* de $\mathcal{B}$ à $\mathcal{A}$ est un algorithme B^{O_A} pour résoudre $\mathcal{B}$, qui a accès à un *oracle* O_A de résolution de $\mathcal{A}$

- Oracle : « boîte noire »

Notation : $\mathcal{B} \leq \mathcal{A}$

Coût d'une réduction

Coût de $B^{O_A} \approx$ « coût classique » + nombre d'appels à O_A

- On ne compte (généralement) pas le « coût » de l'oracle

Exemples d'application de réductions

Théorie de la complexité

Comparer les classes de complexités de différents problèmes

- Exemple : B se réduit polynomialement à $3SAT \Rightarrow \mathcal{C}(B) \subseteq NP$

Algorithmique, développement logiciel

Si $\mathcal{B} \leq \mathcal{A}$ *et que la réduction est suffisamment efficace*, on peut concentrer les efforts sur $\mathcal{A}$

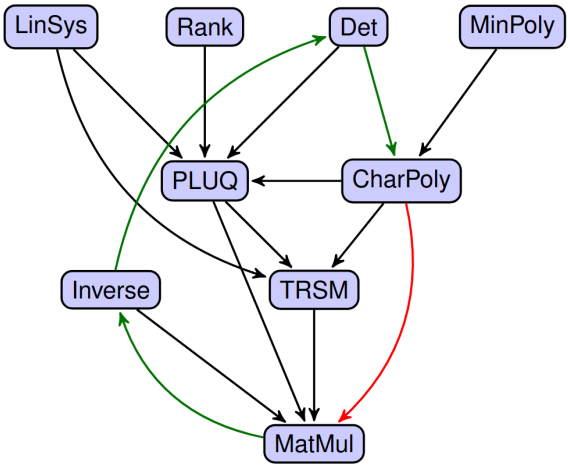
- Exemple : réduire l'algèbre linéaire à la multiplication de matrices

Cryptographie

Montrer que la sécurité d'un système (à formaliser ; pas facile à faire !) se réduit à une hypothèse « plus faible »

- Exemple : $Adv^{\text{IND-CPA}}_{\text{CTR[AES-128]}}(t, q) \leq Adv^{\text{PRP}}_{\text{AES-128}}(t, q') + q'(q' - 1)/2^{127}$

Exemples d'application de réductions



$$\left.\begin{matrix}\text{LinSys}\\ \text{Det}\\ \text{Rank}\\ \text{PLUQ}\\ \text{TRSM}\\ \text{Inverse}\end{matrix}\right\} = O(\text{MatMul})$$

$$\text{MatMul} = O\left(\begin{matrix}\text{Det},\\ \text{PLUQ},\\ \text{CharPoly},\\ \text{Inverse}\end{matrix}\right)$$

$$\text{CharPoly} = O(\text{MatMul})\ ?$$

Figure: Graphe de réductions en algèbre linéaire. Figure par Clément Pernet

Exemples d'application de réductions

Théorie de la complexité

Comparer les classes de complexités de différents problèmes

- Exemple : B se réduit polynomialement à $3SAT \Rightarrow \mathcal{C}(B) \subseteq NP$

Algorithmique, développement logiciel

Si $\mathcal{B} \leq \mathcal{A}$ *et que la réduction est suffisamment efficace*, on peut concentrer les efforts sur $\mathcal{A}$

- Exemple : réduire l'algèbre linéaire à la multiplication de matrices

Cryptographie

Montrer que la sécurité d'un système (à formaliser ; pas facile à faire !) se réduit à une hypothèse « plus faible »

- Exemple : $Adv^{\text{IND-CPA}}_{\text{CTR[AES-128]}}(t, q) \leq Adv^{\text{PRP}}_{\text{AES-128}}(t, q') + q'(q' - 1)/2^{127}$

(Quelques) types de réductions

Réductions « par oracle » / « de Turing » / (non terminales)

Celles ci-dessus

- *Plusieurs* appels à l'oracle possibles
- « post-traitement » possible

Réductions « many-one » / (terminales)

- Un seul appel à l'oracle possible, qui doit être terminal
- (Pertinent surtout pour les problèmes de décision)

Réductions *non-boîtes noires*

Avec un algorithme explicite à la place de l'oracle !!

Réductions *non-boîtes noires*

On résout $\mathcal{B}$ en *transformant* un algorithme **explicite** résolvant $\mathcal{A}$

(Pseudo-)formalisation

Une *réduction* de $\mathcal{B}$ à $\mathcal{A}$ est un algorithme R qui prend en entrée un algorithme A résolvant $\mathcal{A}$ et renvoie un algorithme $B = R(A)$ résolvant $\mathcal{B}$

- « Algorithme » : généralement des *straight-line programs* (SLP) : des circuits

SLP

Programme sans-branchement : chaque étape de l'algorithme effectue un calcul et affecte le résultat dans un registre ; par exemple :

```
type reg = R of int
type 'a opnd = Cst of 'a | Reg of reg
type 'a optn = Add of 'a opnd * 'a opnd | Sub of 'a opnd * 'a opnd
             | Mul of 'a opnd * 'a opnd | Inv of 'a opnd
type 'a step = Step of reg * 'a optn
type 'a slp = 'a step list
```

Intérêt des réductions non-boîte noires

Avantages

- Permettent de contourner des impossibilités/difficultés
- Notamment liées à la taille d'un résultat
- Exemple : réduire (efficacement) l'inversion de matrice (taille de sortie : $n^2|x|$) au calcul du déterminant (taille de sortie : $|x|$) !

Inconvénients

- Peut restreindre le modèle de calcul

Un exemple : principe de transposition

Théorème

Soit $\mathbf{M}_A$ une matrice $\in \mathbb{K}^{n \times m}$, A un SLP calculant $\mathbf{v} \mapsto \mathbf{v} \cdot \mathbf{M}_A$ en c opérations sur $\mathbb{K}$, on peut construire un SLP B calculant $\mathbf{v} \mapsto \mathbf{v} \cdot \mathbf{M}_A^t$ (ou de façon équivalente, $\mathbf{v} \mapsto \mathbf{M}_A \cdot \mathbf{v}$) en $c + \alpha(n - m)$ opérations sur $\mathbb{K}$ (avec $\alpha \leq 3$)

Arguments de preuve

- Approche 1 : (si les registires sont en écriture unique) on peut définir un DAG sous-tendant A, et le SLP B s'obtient en le transposant (!)
- Approche 2 : par induction sur le SLP d'instructions réduites :
  - $R_i \leftarrow\!\shortmid R_i + R_j \quad \Rightarrow \quad R_j \leftarrow\!\shortmid R_j + R_i$
  - $R_i \leftarrow\!\shortmid R_i \times \lambda \quad \Rightarrow \quad R_i \leftarrow\!\shortmid R_i \times \lambda$
  - $S_1; S_2 \quad \Rightarrow \quad S_2; S_1$

Référence

Pour un peu plus de détails (mais pas tant de preuve) : AECF

Table des matières

Différentiation automatique

Théorème

Soit A un SLP calculant l'évaluation $(a_1, a_2, \ldots, a_n) \mapsto f(a_1, a_2, \ldots, a_n)$ d'une fraction rationnelle $f \in \mathbb{K}(x_1, x_2, \ldots, x_n)$ en c opérations sur $\mathbb{K}$, on peut construire un SLP B calculant A **et** $(a_1, a_2, \ldots, a_n) \mapsto \{(\partial f/\partial x_i)(a_0, a_1, \ldots, a_n)\}_{1 \leq i \leq n}$ en $\leq 5c$ opérations sur $\mathbb{K}$

- Le surcoût *en temps* est **constant** en n!!

Corollaire

Soit A un SLP calculant le déterminant de $\mathbf{M} \in \mathbb{K}^{n \times n}$ en c opérations sur $\mathbb{K}$, on peut constuire un SLP inversant $\mathbf{M}$ (quand inversible) en $\leq 5c + n^2$ opérations sur $\mathbb{K}$

Preuve du corollaire

$$\mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})} \left[\frac{\partial \det}{\partial \mathbf{A}_{j,i}}(\mathbf{M}) \right]$$

Remarque

L'algorithme de construction du théorème est similaire au « transposé » des algorithmes de *backpropagation*

(Esquisse de) Preuve du théorème (1/2)

Preuve Reprise de :

- *How to compute fast a function and all its derivatives* A variation on the theorem of Baur-Strassen, Jacques Morgenstern, 1985

Approche : induction sur les SLPs, avec les registres comme indéterminées

Convention (w.l.o.g.) : une opération sur R_i ne peut dépendre que de registres $R_{\leq i}$ ou de constantes

Contrainte : un registre n'est jamais utilisé deux fois pour une affectation (pas d'impact sur le coût *temporel*)

Cas de base

- $\partial R_i/\partial R_j$ est la fonction constante $1_{\mathbb{K}}$ si $j = i$, $0_{\mathbb{K}}$ sinon

Cas inductif

- Soit $S = (R_h \leftarrow op_h) :: S_{\text{tail}}$ un SLP de n registres d'entrée $(R_1, \ldots, R_n)$ et t étapes
- Alors S_{tail} SLP de $n+1$ registres d'entrées $(R_1, \ldots, R_n, R_h)$, $t-1$ étapes
- $S[R_1 := v_1, \ldots, R_n := v_n]$ et $S_{\text{tail}}[R_1 := v_1, \ldots, R_n := v_n, R_h := op_h(v_1, \ldots, v_n)]$ calculent la même fonction

(Esquisse de) Preuve du théorème (2/2)

Cas inductif (suite)

- Par induction, S_{tail} et ses $n+1$ dérivées partielles se calculent pour au plus 5 fois le coût de S_{tail} seule
  - Donne $S_{\text{tail}}(v_1, \ldots, v_n, v_h)\,; (\partial S_{\text{tail}}/\partial R_1)(v_1, \ldots, vn, v_h)\,; \ldots\,; (\partial S_{\text{tail}}/\partial R_h)(v_1, \ldots, v_n, v_h)$
- Il suffit de montrer que S et ses n dérivées partielles peuvent alors se calculer en au plus 5 opérations sur $\mathbb{K}$:
  - Formule magique (règle de la chaîne):

$$\begin{aligned}(\partial S/\partial R_i)(v_1, \ldots, v_n) =& (\partial S_{\text{tail}}/\partial R_i)(v_1, \ldots, v_n, v_h) \; + \\ & (\partial S_{\text{tail}}/\partial R_h)(v_1, \ldots, v_n, v_h) \times (\partial op_h/\partial R_i)(v_1, \ldots, v_n)\end{aligned}$$

  - Et $(\partial op_h/\partial R_i)$ est la fonction nulle sauf pour au plus deux $R_i's$!
  - Un des cas les plus coûteux : $op_h = R_i \times R_j \Rightarrow \partial op_h/\partial R_i = R_j \Rightarrow (\partial S/\partial R_i)(v_1, \ldots, v_n) = (\partial S_{\text{tail}}/\partial R_i)(v_1, \ldots, v_n, v_h) + (\partial S_{\text{tail}}/\partial R_h)(v_1, \ldots, v_n, v_h) \times v_j$: coût 2, pareil pour $(\partial S/\partial R_i)(v_1, \ldots, v_n)$

Implémentation

Implémentation récursive

Il « suffit » d'appliquer l'induction :

- Une récursion (terminale) pour calculer la valeur stockée en R_i (le résultat en R_i de l'évaluation du SLP en $R_{<i}$)
- Couplée à une récursion (non terminale) ; post-traitement : on dépile la valeur stockée en R'_i (l'évaluation en $R_{<i}$ du SLP dérivé suivant R_i) pour mettre à jour les valeurs des $R'_{<i}$

Remarque : à chaque étape récursive, on augmente de un le nombre d'indéterminées suivant lesquelles dériver, mais cela ne nuit pas à l'efficacité de l'algorithme

Try it now!