COURS 26

PRINCIPE DE TRANSPOSITION DE TELLEGEN ET APPLICATIONS

RÉSUMÉ. Le principe de transposition est un ensemble de règles de transformation pour les programmes calculant des applications linéaires. Si P est un programme qui calcule l'application linéaire $x \mapsto Ax$, alors ces règles de transformation permettent d'obtenir un programme P^t qui calcule l'application $y \mapsto A^t y$, où A^t est la matrice transposée de A. En outre, le nombre d'instructions du programme modifié est essentiellement égal au nombre d'instructions du programme initial.

1. INTRODUCTION

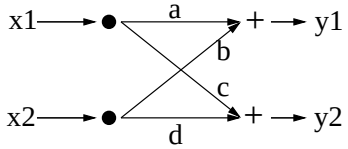
Le *théorème de transposition de Tellegen* affirme que, étant donnée une matrice $\mathbf{M}$ de type $\mathsf{m} \times \mathsf{n}$ à coefficients dans un corps $\mathbb{K}$, sans lignes ni colonnes nulles, tout algorithme qui calcule le produit matrice-vecteur $\mathbf{Mv}$ en L opérations de $\mathbb{K}$ peut être transformé en un algorithme qui calcule le produit $\mathbf{M^t w}$ de la matrice transposée par un vecteur, en utilisant $\mathsf{L} - \mathsf{n} + \mathsf{m}$ opérations dans $\mathbb{K}$. Parfois, par abus de langage, on appelle *principe de Tellegen* l'ensemble des régles de transformation réalisant le passage de l'algorithme direct à l'algorithme dual.

Motivation. Si l'algorithme utilisé pour la multiplication matrice-vecteur est l'algorithme naïf, cet enoncé devient trivial. En effet, dans ce cas :

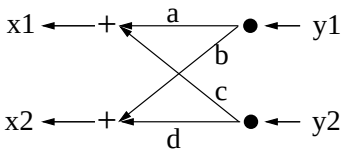
Mv nécessite $\sum_i(\alpha_i - 1) = E - m$ opérations $\pm$ et E opérations $\times$
$M^t w$ nécessite $\sum_j(\beta_j - 1) = E - n$ opérations $\pm$ et E opérations $\times$

où α_i (resp. β_j) est le nombre d'éléments non-nuls de la i^{e} ligne (resp. dela j^{e} colonne) de M et E est le nombre d'entrées non-nulles de M. Par conséquent, pour une matrice complètement générique, le théorème de Tellegen ne présente aucun intérêt. Par contre, si la matrice A admet une structure, il est concevable que l'on dispose d'un algorithme plus rapide que la version naïve pour la multiplier par un vecteur. En utilisant le principe de transposition, on obtient aussitôt un algorithme de complexité analogue pour multiplier A^t par un vecteur.

Considérons l'exemple suivant, qui illustre ce principe en utilisant la représentation par graphes des programmes linéaires. L'algorithme direct prend les valeurs x_1 et x_2 en entrée et renvoie $y_1 = ax_1 + bx_2$ et $y_2 = cx_1 + dx_2$ en sortie. Les arêtes représentent des multiplications par les valeurs constantes a, b, c, d.



Transposer cet algorithme revient à inverser le flot du calcul, c'est-à-dire, inverser le sens des flèches, échanger les $+$ par les $\bullet$ et échanger les entrées et les sorties. L'algorithme ainsi obtenu prend y_1, y_2 en entrée et renvoie $x_1 = ay_1 + cy_2$ et $x_2 = by_1 + dy_2$. Il calcule donc bien l'application transposée de l'application initiale. De plus, le nombre d'opérations arithmétiques utilisées par les deux algorithmes est le même : 4 multiplications et 2 additions.



Historique. Cette technique de transformation des programmes linéaires est issue du domaine des circuits électroniques [28, 5, 23, 10, 1] et en théorie du contrôle [16]. Le principe même remonte aux années 50 ; on en trouve les traces dans un article de Tellegen [28] sur les réseaux électriques. Il a été généralisé aux filtres digitaux par Fettweis [10], où l'on trouve une version embryonnaire de la version par graphes. Le théorème de Tellegen a été redémontré plusieurs fois, dans divers contextes et degré de généralité, par Fiduccia [11], Hopcroft et Musinski [15] par Knuth et Papadimitriou [19] et par Kaminski, Kirkpatrick et Bshouty [18]. En calcul formel, il a été popularisé dans les travaux de Ben-Or, Tiwari [3], Kaltofen, Canny, Lakshman [9, 17], Shoup [24, 25, 26], Lecerf, Schost [20], Hanrot, Quercia, Zimmermann [14], Zippel [30], von zur Gathen et Gerhard [12], ... Il reste parfois peu connu, comme le montre cet extrait de l'article [29] de Wiedemann : "I was not able to find an example of a linear operator that is easy to apply but whose transpose is difficult to apply".

Utilité du principe de Tellegen. Beaucoup de problèmes en calcul formel s'expriment en termes d'algèbre linéaire structurée (multiplication par un vecteur ou résolution de système). Par exemple, les opérations élémentaires sur les polynômes (multiplication, division, évaluation-interpolation, extrapolation, etc.) sont *linéaires*, en ce sens qu'ils peuvent se coder en termes Mv, où M est une matrice structurée (Toeplitz, compagnon, Vandermonde, Cauchy,...)

Grâce au principe de Tellegen, comprendre, analyser et améliorer un algorithme signifie également comprendre, analyser et améliorer son transposé. Le principe de Tellegen a deux utilités principales : trouver des solutions algorithmiques de meilleure complexité et clarifier le statut de certains algorithmes existant dans la littérature, qui se trouvent être, en fait, les transposés d'algorithmes bien connus. Il permet ainsi le traitement algorithmique unifié des problèmes duaux. On peut résumer l'utilité en disant qu'il divise par deux le nombre d'algorithmes qui reste à découvrir.

Dans la littérature du calcul formel, c'est classiquement son caractère de théorème d'existence qui est exploité : connaissant un algorithme pour une certaine opération linéaire, on en déduit l'existence d'un algorithme pour l'opération duale, de même complexité. Un autre fait, mis en évidence plus récemment dans [7], est que la mise en pratique du principe de Tellegen est bien possible et peut se faire de manière systématique et (quasi-)automatique. Informellement parlant, on pourrait dire que les programmes sont directement transposés, d'une manière similaire à celle utilisée en différentiation automatique [13], mais en tirant profit des spécificités linéaires. Par ailleurs, lorsqu'un problème linéaire doit être résolu, la démarche adoptée consiste à comprendre son problème dual, pour lequel on peut espérer trouver un algorithme rapide. Si tel est le cas, en re-transposant ce dernier, une solution rapide pour le problème de départ est également obtenue.

Exemple. Considérons le problème de l'évaluation d'un polynôme P de degré n en une valeur a. C'est une opération linéaire sur les coefficients de P, de matrice $\mathbf{M} = [1, a, \ldots, a^n]$ dans les bases canoniques. En regardant la transposée de $\mathbf{M}$, on déduit aisément que le problème transposé est le suivant : pour une valeur donnée x_0, calculer les produits $a^i x_0$, pour $0 \leq i \leq n$. Pour ce problème, un algorithme naturel consiste à multiplier x_0 par a, ensuite multiplier le résultat par a, et ainsi de suite.

Comment obtenir le transposé de cet algorithme ? En simplifiant, la réponse est que l'on n'a qu'à parcourir l'algorithme direct en sens inverse, échanger les entrées et les sorties, puis remplacer chaque instruction par sa *transposée*, obtenue en appliquant

un nombre très restreint de règles syntaxiques. Dans ce processus, les boucles `for` montantes deviennent des boucles `for` descendantes.

De cette manière, on obtient *automatiquement* un algorithme pour le problème de départ, à savoir, l'évaluation de P sur a. Dans notre cas, il s'avère que l'algorithme transposé coïncide avec la fameuse *méthode de Horner*, voir la Figure 1. Enfin, on peut se demander pourquoi l'algorithme transposé utilise n opérations de plus que l'algorithme direct. Cette perte s'explique par le théorème de Tellegen : il s'agit tout simplement de la différence entre le nombre de colonnes et le nombre de lignes de $\mathbf{M}$.

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0]$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
  $p_j \leftarrow p_{j-1}$;
  $p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$

Input $p = [p_0, \ldots p_n]$.
for j **from** n **downto** 1 **do**
  $p_j \leftarrow a p_j$;
  $p_{j-1} \leftarrow p_j + p_{j-1}$;
$x_0 \leftarrow p_0$;
Output x_0.

FIG. 1. Le schéma de Horner (à droite) obtenu en transposant l'algorithme qui résout le problème dual.

2. La version en termes de graphes du principe de Tellegen

Dans cette section nous donnons une version du théorème de Tellegen dans un modèle particulier, celui des graphes de calcul (DAG).

Par définition, un DAG $\mathbb{K}$-linéaire est un graphe orienté acyclique $G = (V, E)$ muni d'une fonction de poids $\lambda : E \to \mathbb{K}$. Soit $I = \{x_1, \ldots, x_n\}$ les nœds d'entrée. À tout sommet $v \in V$ on associe une forme linéaire dans $\mathbb{K}[X_1, \ldots, X_n]$ de la façon suivante :
– si $v = x_i \in I$, alors $h_v := X_i$,
– si $v \in V \setminus I$, alors $h_v := \sum_{e=(w,v)\in E} \lambda(e)\, h_w$.
On que dit G calcule la matrice $M = [a_{ij}]$ de type $m \times n$ si les formes linéaires associées aux nœds de sortie sont $F_i = \sum_{j=1}^{n} a_{ij} X_j$, $i = 1, \ldots, m$. Le coût de G est le nombre d'opérations linéaires $c(G)$ induites par G. Avec ces notations, le principe de Tellegen s'ennonce comme suit.

Théorème 1. Soit G un K-DAG qui calcule une matrice $\mathbf{M}$ de type $\mathsf{m} \times \mathsf{n}$. Alors, le graphe transposé $\mathsf{G^t}$ (obtenu en inversant le sens des flèches sans changer leurs poids) calcule la matrice $\mathbf{M^t}$. De plus,

$$c(\mathsf{G}^t) = c(\mathsf{G}) - \mathsf{n} + \mathsf{m}.$$

Esquisse de preuve. La forme linéaire calculée par le sommet v de G est

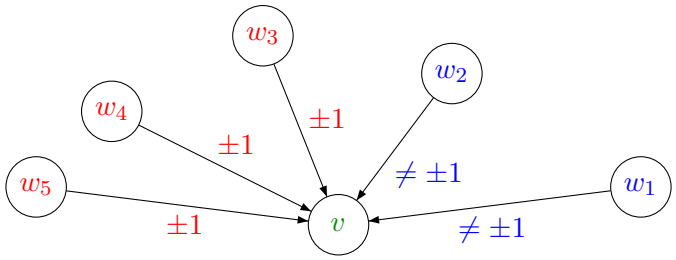
$$h_v = \sum_{j=1}^{n} \left(\sum_{p \in \text{ Chemin}(x_j, v)} \lambda(p) \right) X_j, \quad \text{où} \quad \lambda(p) = \prod_{e \text{ arête } p} \lambda(e).$$

On a donc l'égalité $m_{ij} = \sum_{p \in \text{ Chemin}(x_j, F_i)} \lambda(p)$, qui montre que G^t calcule bien M^t.
L'égalité entre les coûts se déduit du fait que la quantité $c(G) - |I(G)|$ ne dépend pas de l'orientation de G, comme montré par le lemme suivant. □

Lemme 1 (formule pour le coût d'un graphe de calcul)**.** *Avec les notations précédentes, on a l'égalité*

$$c(G) = \big|\{e \in E \mid \lambda(e) \neq \pm 1\}\big| + |E| - |V| + |I|.$$

Démonstration. Tout sommet v qui reçoit $d(v) > 0$ arêtes contribue au coût de G avec $\big|\{e = (w, v) \mid \lambda(e) \neq \pm 1\}\big| + d(v) - 1$ opérations dans $\mathbb{K}$.



La conclusion se déduit alors aisément de la suite d'égalité

$$\sum_{v \in V \setminus I} \big(d(v) - 1\big) = \sum_{v \in V \setminus I} d(v) - \sum_{v \in V \setminus I} 1 = |E| - (|V| - |I|).$$

□

3. Principe de Tellegen pour les programmes linéaires

Le but de cette section est de décrire le principe de Tellegen dans le modèle des programmes linéaires sur des machines à allocation de registres (machines RAM).

3.1. **Machines à registres.** Commençons par définir les machines RAM à instructions linéaires. On fixe un entier M ; il représente la mémoire disponible, de sorte que l'on peut accéder à M registres $R_1, \ldots, R_M$. On y stocke des nombres, c'est-à-dire des éléments du corps $\mathbb{K}$.
Un programme linéaire est la donnée des objets suivants :
– un sous-ensemble $R_{i_1}, \ldots, R_{i_n}$ des registres que l'on appelle *entrée* ;
– un sous-ensemble $R_{o_1}, \ldots, R_{o_m}$ des registres que l'on appelle *sortie* ;
– une suite d'instructions portant sur ces registres, choisies parmi :
  – $R_i = \pm R_j \pm R_k$, avec $1 \leq i, j, k \leq M$,
  – $R_i = \lambda R_j$, avec $1 \leq i, j \leq M$ et λ dans $\mathbb{K}$.

Le fonctionnement d'un tel programme est le suivant : à l'initialisation, les registres d'entrée reçoivent des valeurs $X_1, \ldots, X_n$ dans $\mathbb{K}$ et les autres sont mis à zéro. On effectue ensuite toutes les instructions, puis le programme renvoie les valeurs des registres de sortie. Noter que l'on calcule bel et bien une fonction linéaire des X_i.

3.2. **Transposition de programme.** Nous décrivons maintenant le principe de transposition des programmes linéaires sur une machine RAM.

3.2.1. *Cas d'un jeu d'instructions réduit.* Pour commencer, on traite le cas d'une machine avec un jeu d'instructions limité par rapport au cas général. On ne s'autorise que les instructions du type :
– $R_i = R_i \pm R_j$, avec $1 \leq i, j \leq M$;
– $R_i = \lambda R_i$, avec $1 \leq i \leq M, \lambda \in \mathbb{K}$.
Pour « transposer » un programme comportant des instructions de ce type, on interprète chacune de ces intructions comme une application linéaire $\mathbb{K}^M \to \mathbb{K}^M$.
Pour motiver ce procédé, considérons un exemple. Avec $M = 3$, l'instruction $R_1 = R_1 + R_3$ s'interprète comme l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

R_2 et R_3 n'ont pas changé et R_1 devient $R_1 + R_3$. La transposée de cette application linéaire a pour matrice

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} ;$$

on en déduit que l'on peut la traduire par l'instruction $R_3 = R_3 + R_1$.
De manière générale, la transposée de l'instruction $R_i = R_i + R_j$ est l'instruction $R_j = R_j + R_i$. Par écriture matricielle, on voit que l'instruction $R_i = \lambda R_i$ est inchangée par transposition, et que $R_i = R_i - R_j$ se transpose en $R_j = R_j - R_i$.

On peut alors définir le programme transposé P^t d'un programme P :
– les entrées de P^t sont les sorties de P ;
– les sorties de P^t sont les entrées de P ;
– les instructions de P^t sont les transposées des instructions de P, prises en sens inverse. La dernière condition, le retournement de la liste des instructions, traduit l'égalité $(AB)^t = B^t A^t$, pour des matrices A, B. Cette définition montre que si P calcule une application linéaire $\mathbb{K}^n \to \mathbb{K}^m$, alors P^t calcule bien l'application transposée $\mathbb{K}^m \to \mathbb{K}^n$.

Exemple. Considérons le programme

```
R4:=R4+R2;
R3:=3*R3;
R4:=R4+R3;
R2:=2*R2;
R3:=R3+R2;
R3:=R3+R1;
```

dont les entrées sont R_1, R_2, R_3 et les sorties R_3, R_4. Ce programme calcule l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 3 \end{bmatrix}.$$

Le programme transposé s'écrit

```
R1:=R1+R3;
R2:=R2+R3;
R2:=2*R2;
R3:=R3+R4;
R3:=3*R3;
R2:=R2+R4;
```

et on vérifie qu'il calcule l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 3 \end{bmatrix}.$$

3.2.2. *Cas général.* Le cas du jeu d'instructions général se ramène au cas du jeu d'instructions réduit de manière immédiate. Ainsi, l'instruction $R_1 = R_2 + R_3$ est équivalente à la suite d'instructions

```
R1=0
R1=R1+R2
R1=R1+R3
```

Il est donc possible de le transposer sous la forme

```
R3=R3+R1
R2=R2+R1
R1=0
```

De même, on réécrit aisément l'instruction $R_i = \lambda R_j$ en utilisant le jeu d'instructions réduit, ce qui permet de la transposer, ...

3.3. **Optimisations.** Notre principe de transposition n'est pas complètement satisfaisant : au vu des règles de réécriture ci-dessus, il semble que l'on puisse perdre jusqu'à un facteur 3 (en termes de nombre de lignes). On peut optimiser le code produit, afin de regagner tant que possible les lignes perdues, en utilisant pour ce faire les règles suivantes.

Suppression des zéros. La transformation pour passer du cas général au jeu d'instructions réduit introduit des lignes du type $R_i = 0$. On peut supprimer une telle ligne, à condition de réécrire les instructions suivantes en conséquence.

Suppressions des recopies. Après avoir supprimé les zéros, il se peut qu'il reste des lignes du type $R_i = \pm R_j$. On peut également supprimer ce type de ligne, à condition de réécrire de manière judicieuse les instructions qui suivent.
On peut en fait montrer qu'il est possible d'obtenir un code transposé qui fait exactement le même nombre de lignes que l'original pour une matrice « générique ».

4. Applications

Dans cette section, notre but est de démontrer l'utilité pratique du principe de Tellegen. Dans cette optique, nous utilisons les mêmes techniques de transformation de programmes linéaires que dans l'exemple de l'introduction afin d'engendrer des programmes linéaires pour effectuer les opérations suivantes sur les polynômes à une variable : multiplication, division euclidienne, évaluation et interpolation multipoint. Dans la Figure 2 nous donnons une liste (non-exhaustive) d'opérations linéaires de base sur les polynômes à une variable (voir la Figure 2) et leurs problèmes transposés. Tout algorithme résolvant un problème de la colonne de gauche peut être transposé en un algorithme de même complexité pour le problème correspondant de la colonne de droite.

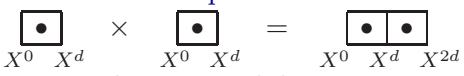
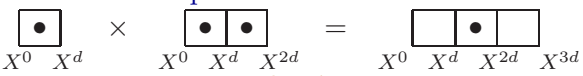
problème direct	problème transposé
multiplication	produit médian
	
division euclidienne	extension de récurrences
$A \mapsto A \mod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
évaluation multipoint	sommes de Newton pondérées
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$
interpolation	décomposition en éléments simples
(systèmes de Vandermonde)	(systèmes de Vandermonde transposés)
décalage de polynômes	évaluation dans les factorielles descedantes
$P(X) \mapsto P(X+1)$	$P = \sum a_i X^{\underline{i}} \mapsto (P(0), \ldots, P(n-1))$
extrapolation sur $0, 1, 2, \ldots$	division modulo $(X-1)^n$
composition modulaire	projection des puissances
$\ldots$	$\ldots$

Fig. 2. Dictionnaire de Tellegen pour les polynômes univariés.

4.1. **Produit médian des polynômes.** Nous commençons par transposer la multiplication des polynômes. Cette opération n'est pas linéaire, mais elle le devient quand on fixe une des deux opérandes. Fixons une fois pour toutes un polynôme f dans $\mathbb{K}[X]$, de degré m. Pour n quelconque, considérons à nouveau l'application de multiplication par f, qui à un polynôme g de degré inférieur à n associe le produit fg, de degré inférieur à $m+n$. La transposée de cette opération consiste à extraire les coefficients de $X^m, X^{m+1}, \ldots, X^{m+n}$ du produit d'un polynôme de degré au plus $m+n$ par le polynôme réciproque $\tilde{f}$ de f.

Par exemple, soit $f = 2 + X + 3X^2$; la matrice de la multiplication $g \mapsto fg$, avec g de degré au plus 2, est donnée par la matrice de type Toeplitz

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 3 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}.$$

Sa transposée est

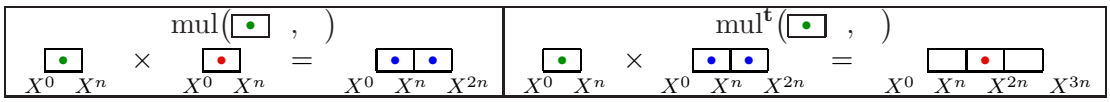
$$\begin{bmatrix} 2 & 1 & 3 & 0 & 0 \\ 0 & 2 & 1 & 3 & 0 \\ 0 & 0 & 2 & 1 & 3 \end{bmatrix}.$$

Cette matrice est la partie médiane de la matrice de Toeplitz suivante

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ \mathbf{2} & \mathbf{1} & \mathbf{3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{3} \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

qui représente le produit du polynôme $\tilde{f} = 3 + X + 2X^2$ par un polynôme h de degré 4. Cette partie médiane donne les coefficients de degré $2, 3, 4$ du produit $\tilde{f}h$. En vertu du principe de Tellegen appliqué aux matrices de Toeplitz, le coût de cette opération est celui du produit fg ; en particulier, si $m = n$, ceci permet d'effectuer le calcul de la partie médiane pour le coût d'une multiplication en degré n, au lieu des deux qu'on attendrait naïvement.

Cette opération est centrale dans toute la suite de ce cours. Elle constitue une brique de base, sur laquelle se fonde une bonne partie des algorithmes qui y sont présentés. Calculer vite des produits transposés est donc un problème important à résoudre avant de passer à la transposition d'autres algorithmes plus compliqués.



Example :

$$\underset{\text{mul}\,(2+3x,\, a+bx)}{\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix}} \qquad \underset{\text{mul}^{\mathbf{t}}\,(3+2x,\, a+bx+cx^2)}{\begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix}}.$$

Il y a plusieurs algorithmes pour multiplier des polynômes, à chacun correspond donc un algorithme de même complexité pour calculer le produit transposé ; cet algorithme transposé peut être obtenu en appliquant le principe de Tellegen, soit dans sa version graphes, soit par transformation de programmes linéaires. Plus exactement, on a le résultat général suivant, obtenu par [14] par une méthode différente.

Théorème 2. *On peut transformer tout algorithme de multiplication polynomiale de coût* $\mathsf{M}(\mathbf{n})$ *en degré* n*, en un algorithme pour le produit médian en degrés* $(n, 2n)$ *de complexité* $\mathsf{M}(\mathbf{n}) + \mathbf{O}(\mathbf{n})$.

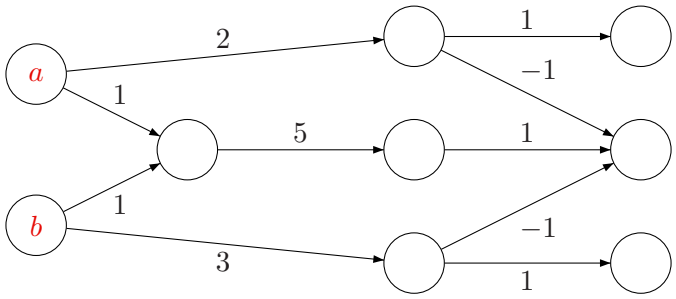
Exercice 1. *Vérifier le théorème dans le cas de la multiplication naïve des polynômes. Expliciter l'algorithme transposé dans ce cas.*

Une application de cette idée permet d'améliorer l'opérateur de Newton pour l'inverse d'une série formelle $a(X) = \sum_{i \geq 0} a_i X^i$, avec $a_0 \neq 0$. Ce calcul repose sur une itération de la forme

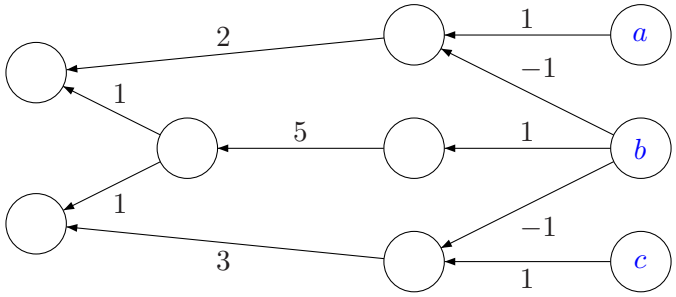
$$b_0 = 1/a_0 \quad \text{et} \quad b_{i+1} = b_i + b_i(1 - ab_i) \bmod X^{2^{i+1}}$$

pour $i \geq 0$. En regardant cette formulation de plus près, on constate qu'il est possible d'utiliser la produit transposé pour accélérer une des multiplications. Enfin, puisque l'inversion d'une série formelle est un point-clé pour la division euclidienne, l'accélération se répercute au niveau de la division.

Exemple : Un DAG qui calcule le produit $(2 + 3x)(a + bx)$ *à la Karatsuba* ; le DAG transposé calcule le produit médian de $3 + 2x$ et $a + bx + cx^2$ *à la Karatsuba*.



$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a + 2b \\ 3b \end{bmatrix} = \begin{bmatrix} 2a \\ 5(a+b) - 2a - 3b \\ 3b \end{bmatrix}$$



$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a + 3b \\ 2b + 3c \end{bmatrix} = \begin{bmatrix} 2(a-b) + 5b \\ 3(b-c) + 5b \end{bmatrix}$$

Remarque 1. *Le produit médian $(n, 2n)$ de A et de B peut être lu*
(i) sur les éléments du produit d'une matrice de Hankel H_B et un vecteur v_A. Ceci implique qu'on peut effectuer un tel produit en $\mathsf{M}(n) + \mathcal{O}(n)$ opérations de $\mathbb{K}$.
(ii) la partie de fort poids de la convolution AB mod $X^{2n} - 1$: en effet, tout matrice de Hankel de type $n \times n$ peut être plongée dans une matrice circulante de type $2n \times 2n$. Ceci implique que dans le modèle de multiplication polynomiale par FFT, un produit médian $(n, 2n)$ peut être effectué un utilisant 3 FFT (2 directe et une inverse) de taille uniquement $2n$ (et non $3n$, comme l'algorithme naïf) Cette interprétation n'est plus utile dans le modèle de multiplication par l'algorithme de Karatsuba. Hanrot, Quercia et Zimmermann [14] semblent être les premiers à avoir conçu un algorithme de type Karatsuba pour le produit médian.

4.2. **Exemple : algorithme de Karatsuba et son transposé.**

mul

Input (c, d).

$e \leftarrow c + d$;
$U \leftarrow \mathrm{mul}(a, c)$;
$V \leftarrow \mathrm{mul}(b, d)$;
$W \leftarrow \mathrm{mul}(a + b, e)$;
$W \leftarrow W - U - V$;

Output (U, V, W).

$\mathrm{mul}^{\mathbf{t}}$

Input (U, V, W).
$V \leftarrow V - W$;
$U \leftarrow U - W$;
$e \leftarrow \mathrm{mul}^{\mathbf{t}}(a + b, W)$;
$d \leftarrow \mathrm{mul}^{\mathbf{t}}(b, V)$;
$c \leftarrow \mathrm{mul}^{\mathbf{t}}(a, U)$;
$c \leftarrow c + e$;
$d \leftarrow d + e$;
Output (c, d).

4.3. **Division avec reste et extension des récurrences à coefficients constants.** On peut montrer que le dual du problème de la division avec reste d'un polynôme de degré N par un polynôme fixé de degré n est l'extension des récurrences linéaires à coefficients constants. Étant donnés les n premiers termes d'une suite qui vérifie une récurrence linéaire à coefficients constants d'ordre n, il s'agit de calculer la tranche des N termes suivants.
Pour une récurrence d'ordre $n = 1$, la preuve est imédiate : étendre une récurrence $u_{n+1} = au_n$, de condition initiale $u_0 = x_0$ revient à calculer $x_0, ax_0, \ldots, a^N x_0$ à partir de x_0. Or, on a vu que le dual de ce problème est l'évaluation polynomiale en a, autrement dit, la division avec reste modulo $X - a$.

Exercice 2. *Finir la preuve dans le cas général (n quelconque).*

Cette dualité permet de clarifier le statut de l'algorithme de Shoup [24, 26] pour l'extension de récurrences, originellement conçu pour éviter le principe de transposition : il est en fait la transposée de l'algorithme de Strassen [27] pour la division avec reste des polynômes.

4.4. **Évaluation multipoint et interpolation.** En calcul formel, une opération importante est l'évaluation multipoint : étant donné un polynôme $P = \sum_{i=0}^{n} p_i X^i$ de degré n, il s'agit de calculer les valeurs que prend P sur un ensemble de points $a_0, \ldots, a_n$. C'est un problème algorithmique intéressant en soi, mais qui trouve une application notoire, en conjonction avec l'interpolation, dans les approches modulaires.

En termes matriciels, l'évaluation multipoint se traduit par un produit matrice-vecteur entre la matrice de Vandermonde associée aux points a_i et le vecteur des coefficients du polynôme P, voir la Figure 3. Ainsi, le problème transposé est la multiplication d'une matrice de Vandermonde transposée par un vecteur, c'est-à-dire, le calcul de sommes de Newton pondérées (on les appelle ainsi car si tous les p_i valent 1, on retrouve les sommes des puissances des a_i).

$$\begin{bmatrix} 1 & a_0 & \cdots & a_0^n \\ 1 & a_1 & \cdots & a_1^n \\ \vdots & \vdots & & \vdots \\ 1 & a_n & \cdots & a_n^n \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} P(a_0) \\ P(a_1) \\ \vdots \\ P(a_n) \end{bmatrix} \text{ et } \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_0 & a_1 & \cdots & a_n \\ \vdots & \vdots & & \vdots \\ a_0^n & a_1^n & \cdots & a_n^n \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} \sum p_i \\ \sum a_i p_i \\ \vdots \\ \sum a_i^n p_i \end{bmatrix}.$$

FIG. 3. L'évaluation multipoint et sa transposée, les sommes de Newton pondérées.

Nous proposons un algorithme rapide pour le calcul de ces sommes de Newton. L'idée est que la série $\sum_{s\geq 0}\Big(\sum_{i=0}^{n} p_i a_i^s\Big)X^{-s-1}$ est **rationelle** de la forme $Q = B/A$, où

$$A = (X-a_0)\cdots(X-a_n) \quad \text{et} \quad B = \sum_{i=0}^{n} p_i \frac{A}{X-a_i}.$$

D'où l'algorithme suivant : on calcule A et B par divide-and-conquer et on retourne les n premiers coefficients de la série $Q = \dfrac{B}{A}$. Par transposition, nous en déduisons un algorithme pour l'évaluation multipoint de même complexité

$$\frac{3}{2}\,\mathsf{M}(n)\log(n) + \mathcal{O}\big(\mathsf{M}(n)\big).$$

Ceci améliore (d'un facteur constant) les algorithmes classiques d'évaluation, dus à Borodin et Moenck [6, 21] et revisités par Strassen [27] et Montgomery [22], voir aussi [12, Section 10.1]. Comme les algorithmes classiques, notre nouvel algorithme repose sur une stratégie diviser pour régner, mais remplace, à chaque niveau de récursion, les divisions avec reste par des produits médians, moins coûteux.

Noter qu'il est également possible d'accélérer (par des facteurs constants) les algorithmes classiques d'interpolation et de son problème dual (la résolution de systèmes de Vandermonde transposés). De plus, les méthodes s'étendent au cas plus général du Théorème des restes chinois.

Exercice 3. *Soit $\mathbb{K}$ un corps et soit n un entier. On considère le problème suivant :*

Étant données les valeurs en $0, 1, \ldots, n-1$ d'un polynôme (inconnu) de $\mathbb{K}[X]$ de degré au plus n, calculer les valeurs prises par ce polynôme en $n, n+1, \ldots, 2n-1$.

Déterminer le probème dual, donner un algorithme de complexité $\mathcal{O}(\mathsf{M}(\mathbf{n}))$ pour ce dernier, et en déduire un algorithme explicite de complexité $\mathcal{O}(\mathsf{M}(\mathbf{n}))$ pour le problème de départ.

4.5. **Bonus : l'évaluation et l'interpolation ont des coûts équivalents.** Un dernier exemple d'application, de nature plus théorique, du théorème de Tellegen est la preuve du fait que les problème d'évaluation multipoint et d'interpolation polynomiale sont équivalents du point de vue de la complexité.

Théorème 3 ([8])**.** *Tout algorithme qui effectue l'évaluation multipoint (resp. l'interpolation) sur une famille fixée de points peut être transformé en un algorithme d'interpolation (resp. d'évaluation multipoint) sur la même famille de points de même complexité, à un nombre constant de multiplications polynomiales près.*

$$\mathsf{I}(n) \in O\big(\mathsf{E}(n) + \mathsf{M}(n)\big) \quad \textit{and} \quad \mathsf{E}(n) \in O\big(\mathsf{I}(n) + \mathsf{M}(n)\big).$$

Esquisse de preuve. Supposons qu'on dispose d'un algorithme $\mathcal{I}$ de complexité $\mathsf{I}(n)$ pour l'interpolation sur les points $a_0, \ldots, a_n$. On va montrer comment en déduire un algorithme $\mathcal{E}$ pour l'évaluation multipoint sur $a_0, \ldots, a_n$. L'idée est d'utiliser la factorisation de [9]

$$\mathbf{V_a} = (\mathbf{V_a^t})^{-1}\mathbf{H_a}, \quad \text{où} \quad \mathbf{H_a} = \begin{bmatrix} n & \sum_i a_i & \cdots & \sum_i a_i^n \\ \sum_i a_i & \sum_i a_i^2 & \cdots & \sum_i a_i^n \\ \vdots & \vdots & & \vdots \\ \sum_i a_i^n & \sum_i a_i^n & \cdots & \sum_i a_i^{2n} \end{bmatrix},$$

qui suggère l'algorithme suivant :

- Appliquer $\mathcal{I}$ pour calculer $T(X) = \prod_i (X - a_i)$, en complexité $\mathsf{I}(n)$;
- Calculer $\mathbf{H_a}$ à partir des sommes de Newton de $T(X)$, en complexité $\mathcal{O}(\mathsf{M}(n))$;
- Calculer $\mathbf{v} = \mathbf{H_a p}$ en $\mathcal{O}(\mathsf{M}(n))$;
- Retourner $\mathbf{V_a p} = (\mathbf{V_a^{-1}})^{\mathbf{t}}\mathbf{v}$ en $\mathsf{I}(n)$ en utilisant l'algorithme transposé de $\mathcal{I}$.

□

Notes

Lien avec la différentiation automatique. Canny et al. [9] ont remarqué que le principe de transposition est lié au *mode inverse* en *différentiation automatique* pour le calcul du gradient d'une fonction. En effet, les entrées de $\mathbf{M^t w}$ sont les dérivées partielles de $\mathbf{v M^t w}$ par rapport aux coordonnées de $\mathbf{v}$. Ce produit n'est autre que $\mathbf{wMv}$, et le théorème de Baur-Strassen [2] montre que l'on peut calculer ces dérivées partielles au prix d'un surcoût linéaire.

Théorie de la complexité bilinéaire. Hopcroft & Musinski ont donné dans [15] une version bilinéaire de l'algorithme de Tellegen. Étant donné un algorithme bilinéaire qui utilise N multiplications pour le problème, noté (m, n, p), de multiplier deux matrices de type $m \times n$ et $n \times p$, il est possible g'engendrer cinq algorithmes duaux, chacun utilisant exactement N multiplications, pour les problèmes (n, m, p), (p, m, n), (m, p, n), (n, p, m) et (p, n, m). Ce résultat admet la conséquence utile suivante : si on peut calculer en N multiplications dans $\mathbb{K}$ le produit de deux matrices quelconques sur $\mathbb{K}$ de formats $m \times n$ et $n \times p$, alors $\omega \leq 3 \log_{mnp}(N)$. Par exemple, Bini et al. [4] ont montré qu'il existe un algorithme (approché) pour multiplier une matrice $(3, 2)$ et une matrice $(2, 2)$ en 10 multiplications, obtenant ainsi l'inégalité $\omega \leq 3 \log_{12}(10) = 2,78$.

Dualité pour l'évaluation des monômes à plusieurs variables. Knuth et Papadimitriou [19] ont montré que si $A = [a_{ij}]$ est une matrice carrée inversible dont les éleéments sont des entiers strictement positifs, alors, partant de $\{X_1, \ldots, X_n\}$, le nombre minimum de multplications pour évaluer les monômes

$$\left\{x_1^{a_{11}} X_2^{a_{12}} \cdots X_n^{a_{1n}},\ X_1^{a_{21}} X_1^{a_{22}} \cdots X_n^{a_{2n}}, \ldots,\ X_1^{a_{n1}} X_2^{a_{n2}} \cdots X_n^{a_{nn}}\right\}$$

est le même que le nombre minimum de multplications pour évaluer les monômes

$$\left\{x_1^{a_{11}} X_2^{a_{21}} \cdots X_n^{a_{n1}},\ x_1^{a_{12}} X_1^{a_{22}} \cdots X_n^{a_{n2}}, \ldots,\ X_1^{a_{1n}} x_2^{a_{2n}} \cdots x_n^{a_{nn}}\right\}.$$

Cet enoncé est un cas particulier du théorème de transposition de Tellegen.

Références

[1] A. Antoniou. *Digital Filters : Analysis and Design*. McGraw-Hill Book Co., 1979.

[2] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22 :317–330, 1983.

[3] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of STOC'88*, pages 301–309. ACM Press, 1988.

[4] Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Inform. Process. Lett.*, 8(5) :234–235, 1979.

[5] J. L. Bordewijk. Inter-reciprocity applied to electrical networks. *Appl. Sci. Res. B.*, 6 :1–74, 1956.

[6] A. Borodin and R. T. Moenck. Fast modular transforms. *Comput. System Sci.*, 8(3) :366–386, 1974.

[7] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *Proceedings of ISSAC'03*, pages 37–44. ACM Press, 2003.

[8] A. Bostan and É. Schost. On the complexities of multipoint evaluation and interpolation. *Theoretical Computer Science*, 329(1–3) :223–235, 2004.

[9] J. Canny, E. Kaltofen, and L. Yagati. Solving systems of non-linear polynomial equations faster. In *Proceedings of ISSAC'89*, pages 121–128. ACM Press, 1989.

[10] A. Fettweis. A general theorem for signal-flow networks, with applications. *Archiv für Elektronik und Übertragungstechnik*, 25(12) :557–561, 1971.

[11] C. M. Fiduccia. On obtaining upper bounds on the complexity of matrix multiplication. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 31–40. Plenum, New York, 1972.

[12] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.

[13] J.-C. Gilbert, G. Le Vey, and J. Masse. La différentiation automatique de fonctions représentées par des programmes. Technical report, RR INRIA 1557, 1991.

[14] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm, I. Speeding up the division and square root of power series. *Applicable Algebra in Engineering, Communication and Computing*, 14(6) :415–438, 2004.

[15] J. Hopcroft and J. Musinski. Duality applied to the complexity of matrix multiplication and other bilinear forms. *SIAM Journal on Computing*, 2 :159–173, 1973.

[16] R. E. Kalman. On the general theory of control systems. *IRE Transactions on Automatic Control*, 4(3) :481–491, 1959.

[17] E. Kaltofen, R. M. Corless, and D. J. Jeffrey. Challenges of symbolic computation : my favorite open problems. *Journal of Symbolic Computation*, 29(6) :891–919, 2000.

[18] M. Kaminski, D. G. Kirkpatrick, and N. H. Bshouty. Addition requirements for matrix and transposed matrix products. *Journal of Algorithms*, 9(3) :354–364, 1988.

[19] Donald E. Knuth and Christos H. Papadimitriou. Duality in addition chains. *Bulletin of the European Association for Theoretical Computer Science*, 13 :2–4, 1981.

[20] G. Lecerf and É. Schost. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, 5(1) :1–10, 2003.

[21] R. T. Moenck and A. Borodin. Fast modular transforms via division. *Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972)*, pages 90–96, 1972.

[22] P. L. Montgomery. *An FFT extension of the elliptic curve method of factorization*. PhD thesis, University of California, Los Angeles CA, 1992.

[23] P. Penfield, Jr., R. Spence, and S. Duinker. *Tellegen's theorem and electrical networks*. The M.I.T. Press, Cambridge, Mass.-London, 1970.

[24] V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *Proceedings of ISSAC'91*, pages 14–21. ACM Press, 1991.

[25] V. Shoup. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation*, 20(4) :363–397, 1995.

[26] V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *Proceedings of ISSAC'99*, pages 53–58, New York, 1999. ACM Press.

[27] V. Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik*, 20 :238–251, 1972/73.

[28] B. Tellegen. A general network theorem, with applications. *Philips Research Reports*, 7 :259–269, 1952.

[29] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on informations theory*, IT-32 :54–62, 1986.

[30] R. Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3) :375–403, 1990.