"Dynamic Programming" und automatisches Differenzieren

Simon Blatt

Sommerschule des Graduiertenkollegs 2006

Gliederung

Was ist "Automatisches Differenzieren"?

Programm

$$x = (t_{1-n}, \ldots, t_0)^T \xrightarrow{F} y = (t_{l+1}, \ldots t_{l+m})^T$$

Was ist F'?

Zusätzliche Struktur

- $t_1, \ldots t_l$ Zwischenergebisse
- t_k hängt nur von t_{1-n}, t_{k-1} ab.
- hängt t_i explizit von t_j ab $\rightarrow c_{i,j} := \frac{\partial t_i}{\partial t_j}$ ist bekannt und wir schreiben $j \prec i$.

Automatisches Differenzieren

Automatisches Differenzieren

= Berechnen von F' aus den $c_{i,j}$ mit Hilfe der Kettenregel

Methoden

- "Forward mode"
- "Backward mode"
- Methode von Newsam und Ramsdell

"Forward mode" als Produkt von Matrizen

"Forward mode"

$$\nabla_\xi t_i = \sum_{j \prec i} c_{i,j} \nabla_\xi t_j$$

$$\overrightarrow{v}_i := (\nabla_\xi t_{1-n}, \ldots, \nabla_\xi t_i, 0, \ldots, 0)^T$$

$$\Rightarrow \overrightarrow{v}_i = A_i v_{i-1} = \left(\prod_{j \prec i} A_{i,j} \right) \overrightarrow{v}_{i-1}$$

$$A_{i,j} = Id_q + c_{i,j} e_i e_j^T$$

Matrixschreibweise

$$F' = Q_m \left[\prod_{i=1}^{l+m} A_i \right] P_n^T$$

$$= Q_m \left[\prod_{i=1}^{l+m} \prod_{j \prec i} A_{i,j} \right] P_n^T$$

- $P_n := (Id_n, 0)$,
- $Q_m := (0, Id_m)$

Beispiel

Programm

$$t_1 = c_{1,-1} \cdot t_{-1} + c_{1,0} \cdot t_0$$

$$t_2 = c_{2,1} \cdot t_1$$

$$t_3 = c_{3,2} \cdot t_2$$

$$t_4 = c_{4,2} \cdot t_2$$

$$t_5 = c_{5,2} \cdot t_2$$

$$t_6 = c_{6,2} \cdot t_2 + c_{6,1}\, t_1$$

Matrizen

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_{1,-1} & c_{1,0} & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_{1-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_{1,-1} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Methoden zur Steigerung der Effizienz

- Umordnen des Produktes, ohne das Endergebins zu ändern (-> Kommutativität).
- Löschen unnötiger Spalten und Zeilen (Pruning).
- Optimales Klammern des Produktes (-> Assoziativität, Dynamisches Programmieren)

Hierbei: Komplexität (= Anzahl der skalaren Multiplikationen) soll minimiert werden. Additionen und Speicherzugriffszeiten werden nicht berücksichtigt.

Gliederung

Hilfsmittel: Computational Graph

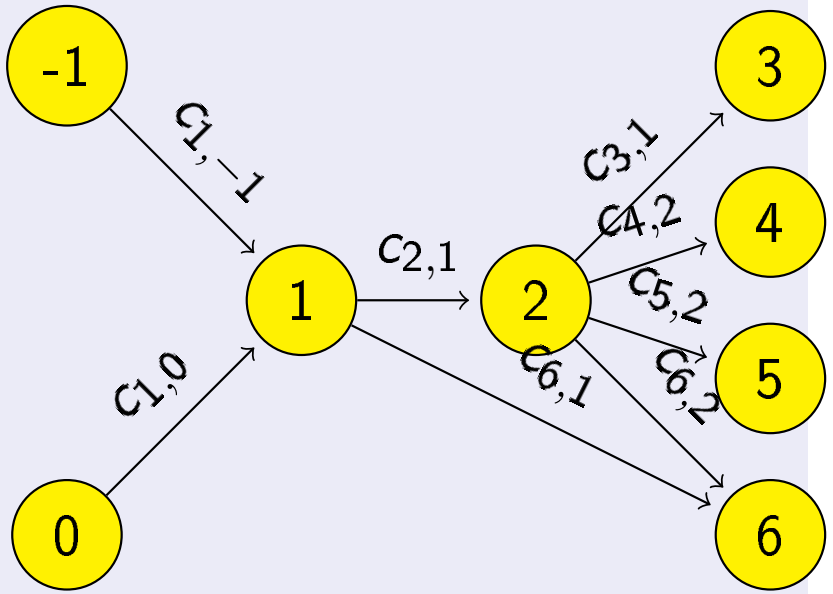
Ecken

$\mathcal{V} := \{1 - n, \ldots l + m\}$.

Kanten

$$\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$$
$$(i, j) \in \mathcal{V} :\Leftrightarrow j \prec i$$

Graph unseres Beispiels:



Umordnen der $A_i \triangleq$ Umsortieren der Ecken

Idee

Relation $\prec$ bleibt erhalten $\Rightarrow$ Neue Reihenfolge liefert dasselbe Ergebnis.

Zulässig sind also Monotone Permutationen:

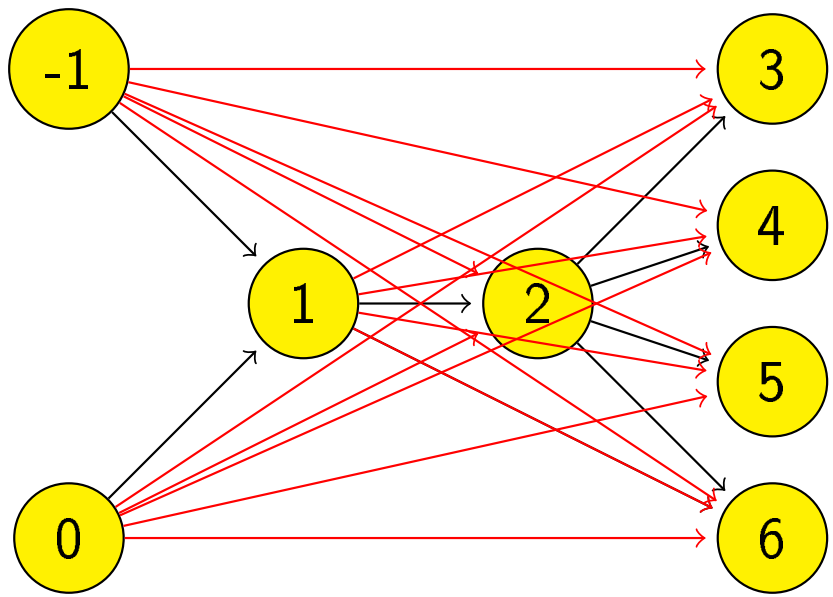
Monotone Permutation

$$\pi \in S(\mathcal{V}) \text{ mit}$$
$$\{j \prec i \Rightarrow \pi(j) < \pi(i)\}$$

Monotone Permutationen erhalten auch den transitiven Abschluss $\prec^+$ der Relation $\prec$:

Transitiver Abschluss

$$j \prec^+ i :\Leftrightarrow j = k_0 \prec \ldots k_r = i$$



RWTH AACHEN UNIVERSITY

Umordnen der $A_{i,j} \triangleq$ Umsortieren der Kanten

Idee

Reihenfolge der Kante entlang eines Weges muss berücksichtigt werden.

Relation zwischen Kanten

$$(j,i) \prec^+ (l,k) :\Leftrightarrow i \prec^+ l$$

Monotone Aufzählung

Aufzählung $(j_k, i_k)_{k=1,\dots e}$, $e = \#\mathcal{E}$, der Kanten mit

$$(j_k, i_k) \prec (j_{\tilde{k}}, i_{\tilde{k}}) \rightarrow k < \tilde{k}$$

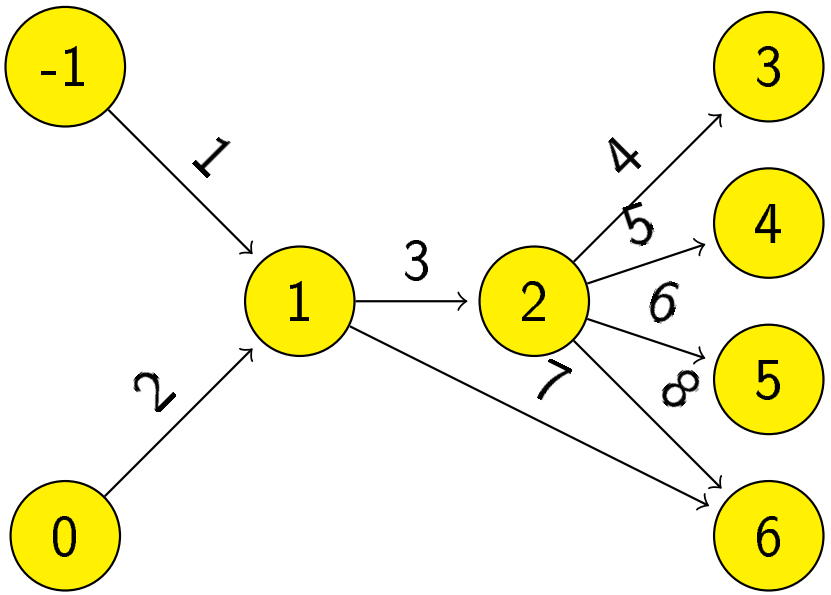
- Es gilt dann

$$F' = Q_m \left[A_{i_e,j_e} \cdots A_{i_1,j_1}\right] P_n^T$$

Beispiele

Lexikographische Ordnung 1

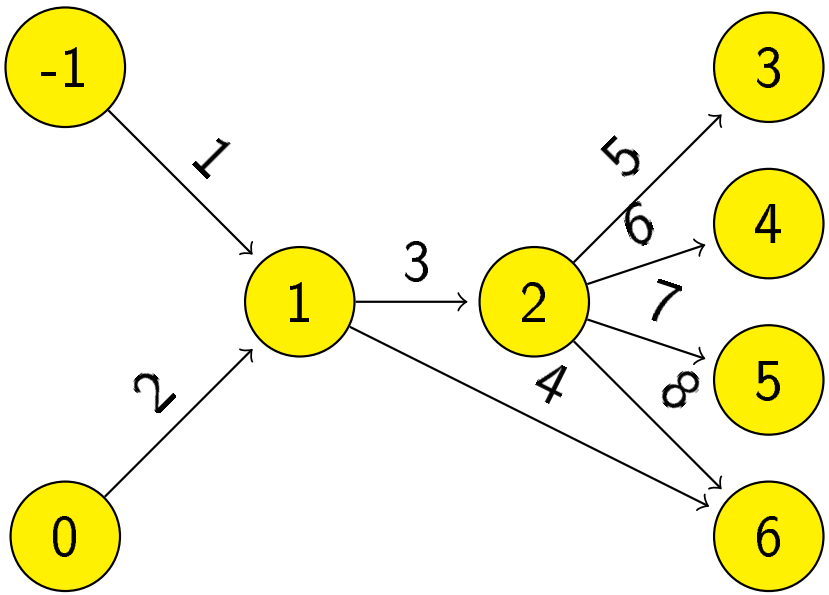
$$k < \tilde{k} :\Leftrightarrow i_k < i_{\tilde{k}} \text{ oder } \left\{ i_k = i_{\tilde{k}} \text{ und } j_k < j_{\tilde{k}} \right\}.$$



$\triangleq$ Forward Mode.

Lexikographische Ordnung 2

$$k < \tilde{k} :\Leftrightarrow j_k < j_{\tilde{k}} \text{ oder } \left\{ j_k = j_{\tilde{k}} \text{ und } i_k < i_{\tilde{k}} \right\}.$$



$\triangleq$ Reverse Mode

Gliederung

Pruning

Modell

$$F' = A_{links} \cdot A_{mitte} \cdot A_{rechts}$$

Frage

Wann hat der Eintrag $(A_{mitte})_{i,j}$ keinen Einfluss auf das Ergebnis?

Beobachtung

- j-te Spalte von A_{links} verschwindet $\Rightarrow$ j-te Zeile von A_{mitte} hat keinen Einfluss auf das Ergebnis.
- i-te Zeile von A_{rechts} verschwindet $\Rightarrow$ i-te Spalte von A_{mitte} ist irrelevant.

Algorithmus anhand eines Beispieles

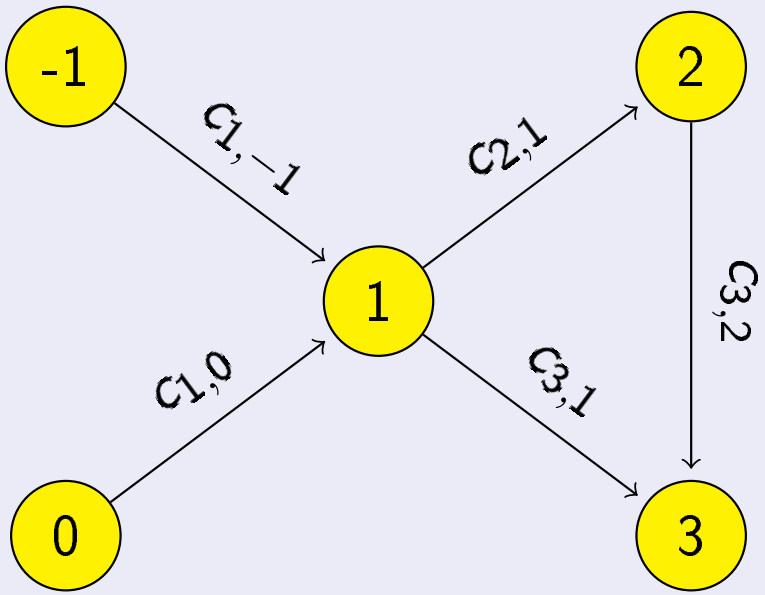
Programm

$$t_1 = c_{1,-1} \cdot t_{-1} + c_{1,0} \cdot t_0$$

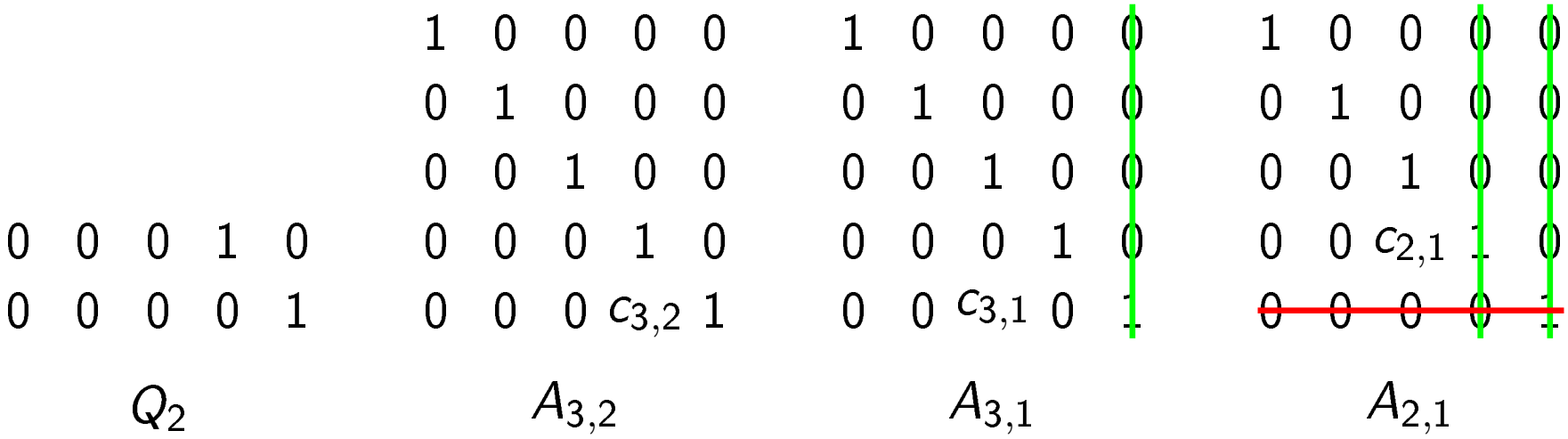
$$t_2 = c_{2,1} \cdot t_1$$

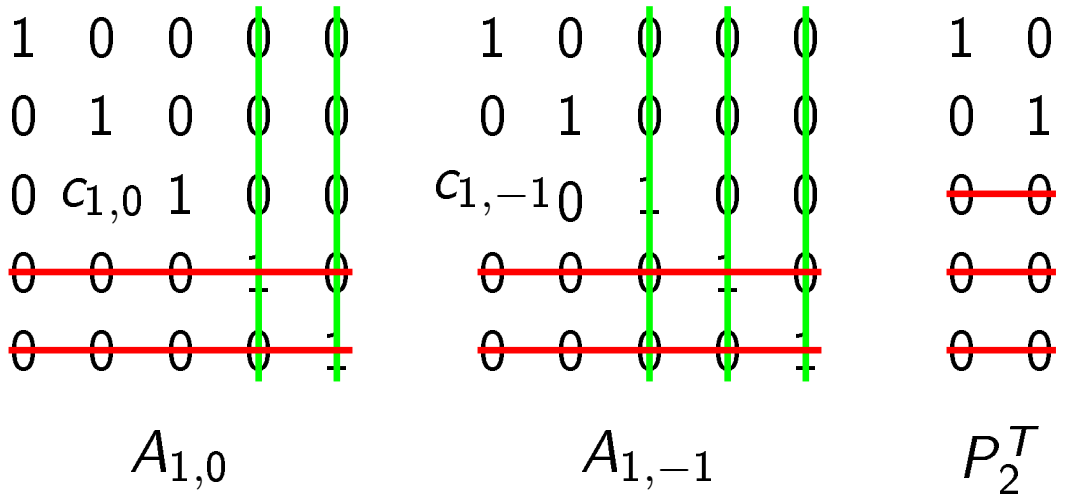
$$t_3 = c_{3,2} \cdot t_2 + c_{3,1}\, t_1$$

Computational graph

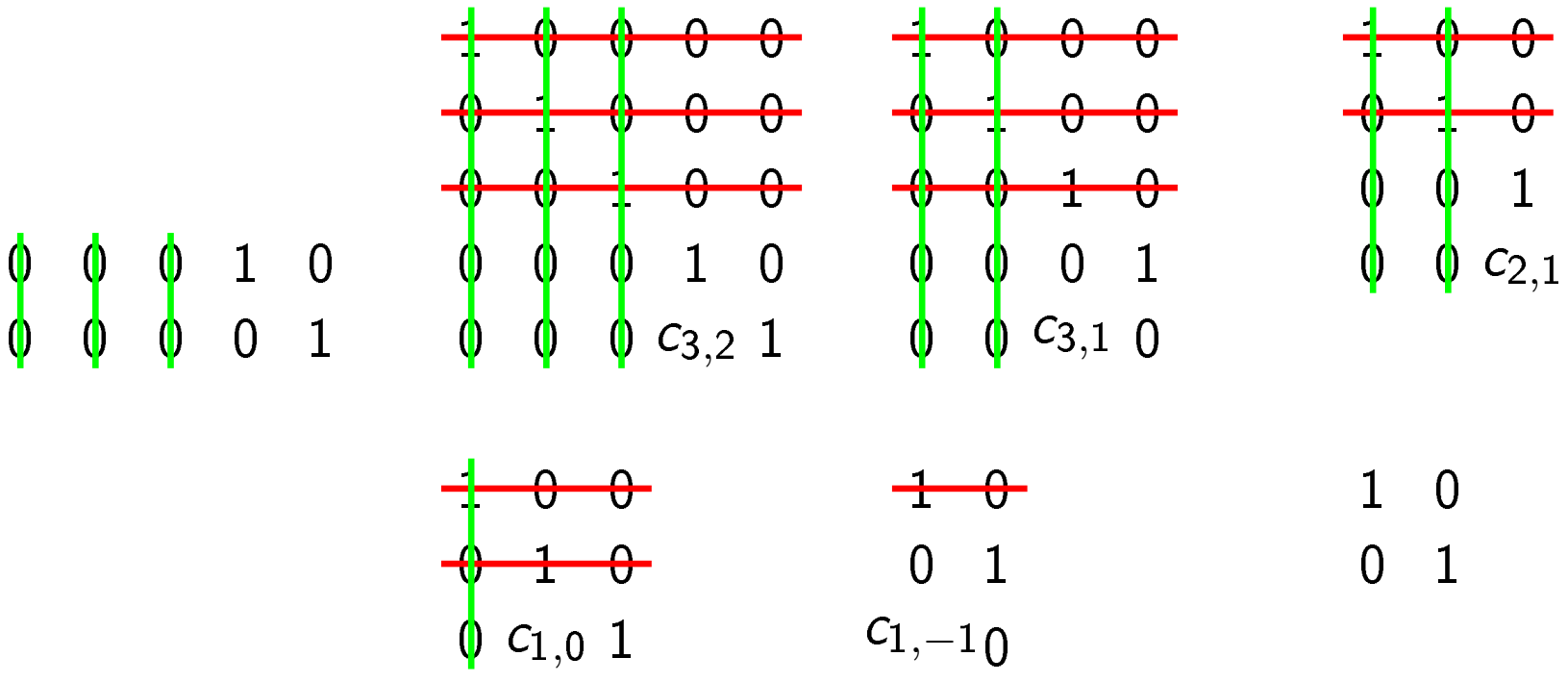


Streichen von rechts nach links





Steichen von links nach rechts



Ergebnis

$$F' = \begin{pmatrix} 1 & 0 \\ c_{3,2} & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ c_{3,1} & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ c_{2,1} \end{pmatrix} \cdot \begin{pmatrix} c_{1,0} & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ c_{1,-1} & 0 \end{pmatrix}$$

Gliederung

Dynamisches Programmieren in der Optimierung

Notwendige Struktur

- Optimale Teilstruktur: Optimal Lösung des Problems ergibt sich aus optimaler Lösung der Teilprobleme.
- Überlappende Teilprobleme: Ein naiver rekursiver Algorithmus löst die Teilprobleme mehrmals.

Unser Problem, ein Matrixprodukt, das aus sehr vielen Faktoren besteht, möglichst gut zu klammern, besitzt diese Struktur:

Matrixprodukte

Problem

Berechnen von $A = \prod_{i=1}^{p} A_i$

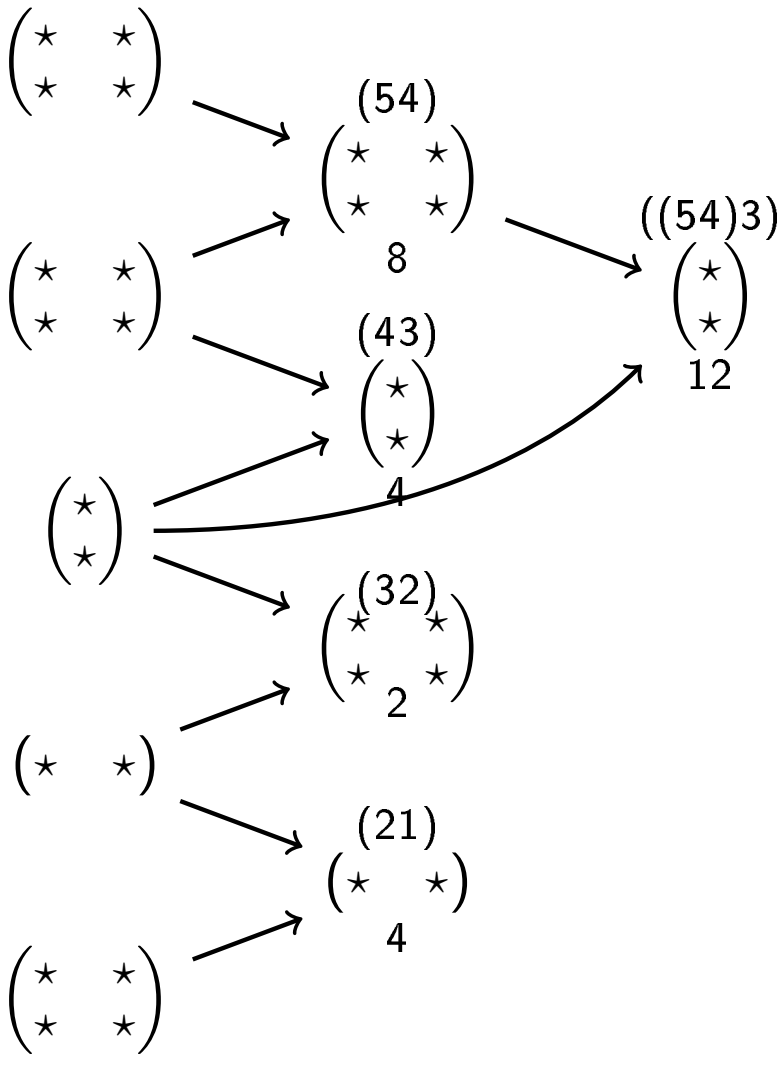
- r_i Anzahl der Zeilen von A_i
- r_{i-1} Anzahl der Spalten von A_i
- $m_{i,j} :=$ minimale Anzahl der Multiplikationen, um das Produkt $A_{i\ldots j} := \prod_{k=i}^{j}$ zu berechnen.

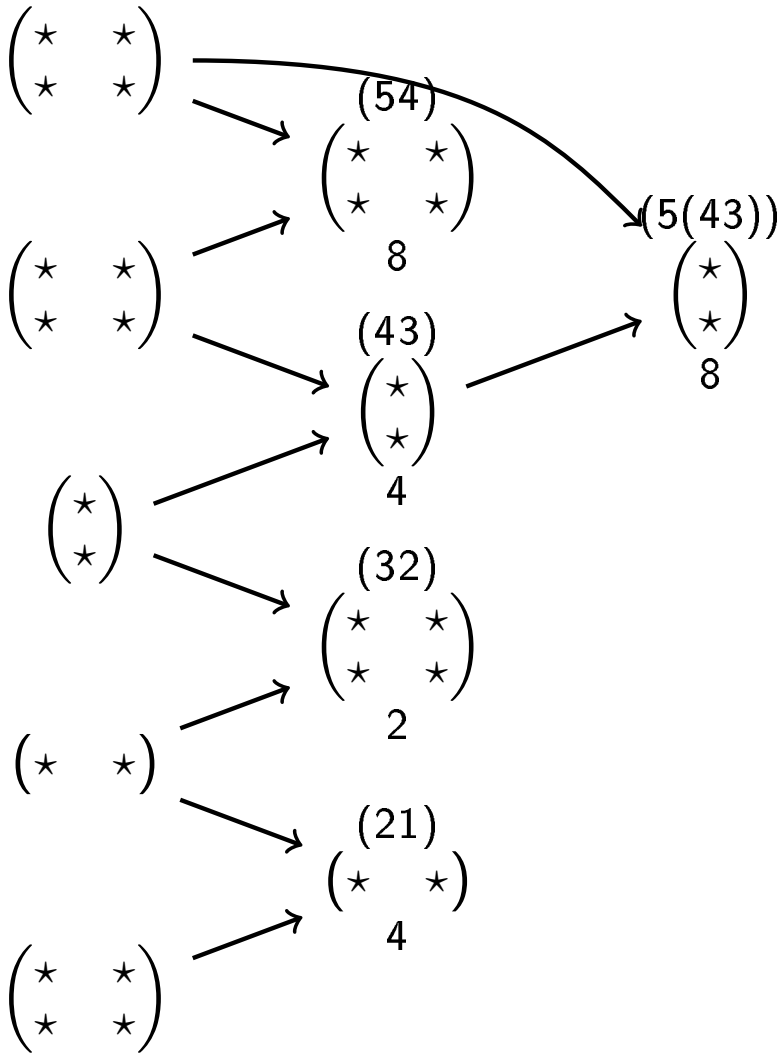
Rekursion

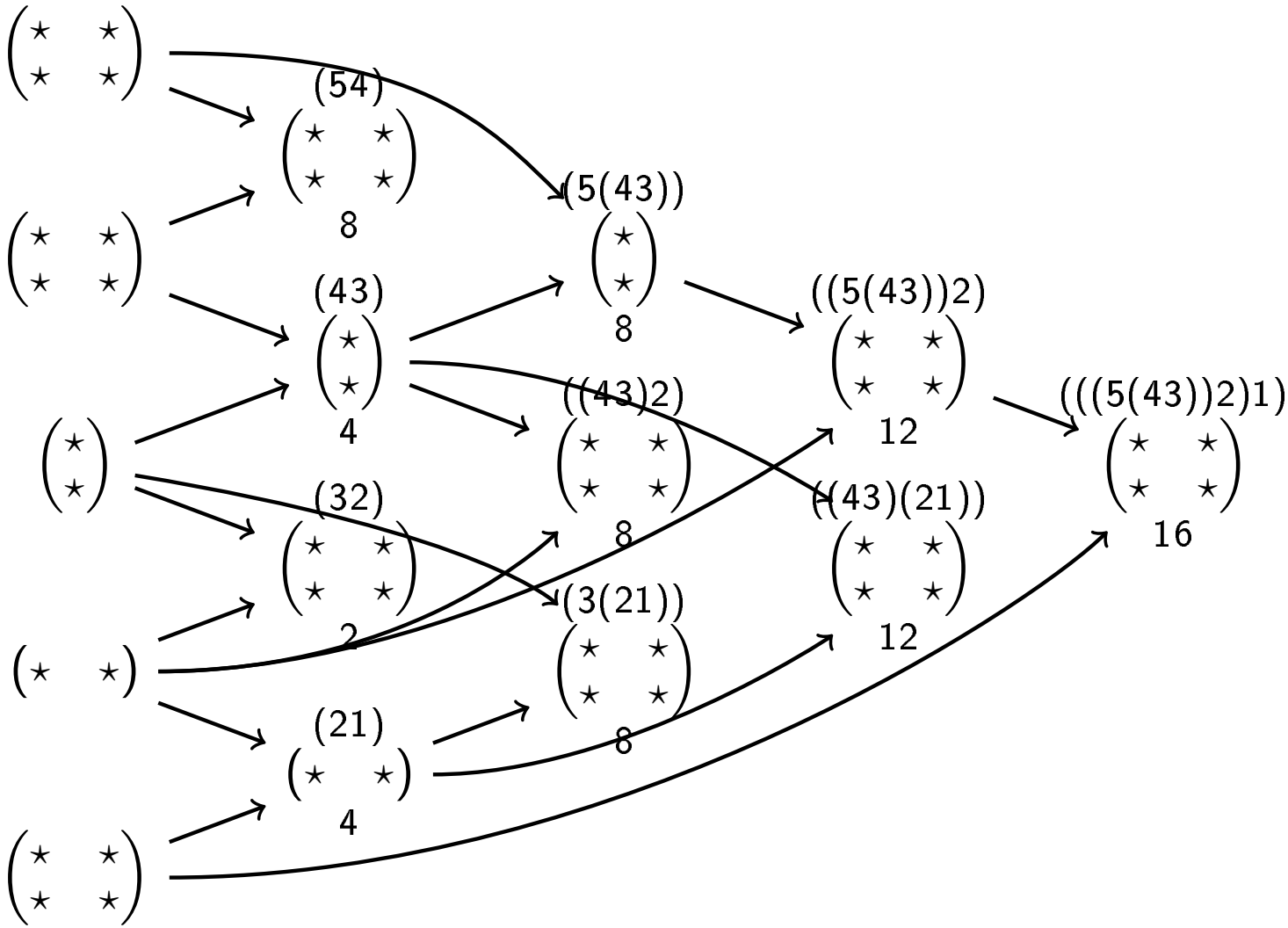
- $m_{i,j} := 0$ falls $i = j$
- $m_{i,j} := \min_{i \le k < j}\{m_{i,k} + m_{k+1,j} + \mu_{ikj}\}$ sonst

- $\mu_{ikj} =$ Anzahl der Operationen, eine $r_i \times r_k$ Matrix mit einer $r_k \times r_{j-1}$ Matrix zu multipizieren $= r_i \cdot r_k \cdot r_{j-1}$.

Algorithmus anhand eines Beispiels







Bemerkungen

- Aufwand ist von der Größenordnung $O(p^3)$.
- Man kann auch noch ausnutzen, dass die A_i in unserem Fall dünnbesetzte Matrizen sind, indem man Multiplikationen mit 0 oder 1 nicht mitzählt.
- Aufwand ist dann von der Größenordnung $O(\nu^3 \cdot p^3)$ (ν maximale Anzahl der Spalten und Zeilen Matrizen).

Gliederung

Pre-bracketing

Aufwand für optimale Klammerung

$$O(p^3)$$

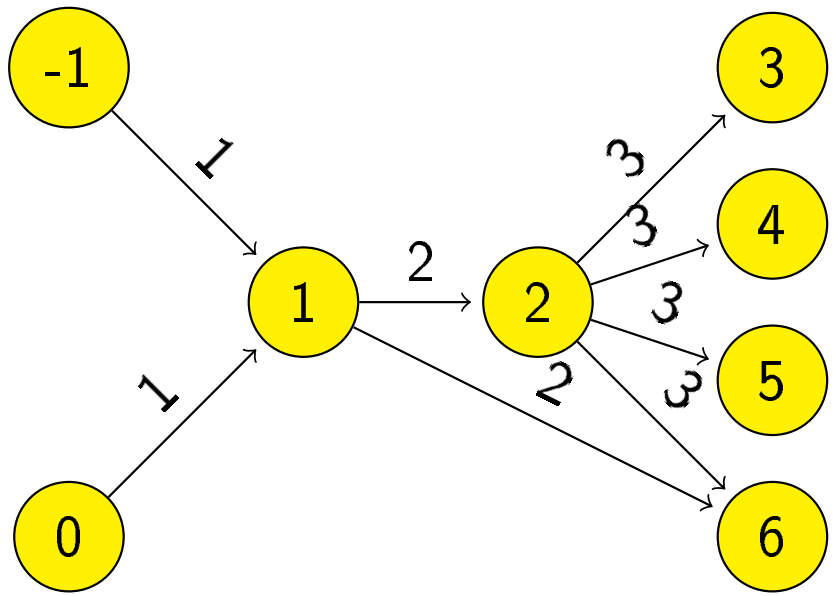
- Sehr aufwendig, falls p groß.

Pre-bracketing

Setzen von Klammern, ohne arithmetische Operationen.

Höhe einer Kante

- $h_{i,j} = 1$, falls $j \leq 0$.
- $h_{i,j} = 1 + \max_{k \prec j} h_{j,k}$



Beobachtung

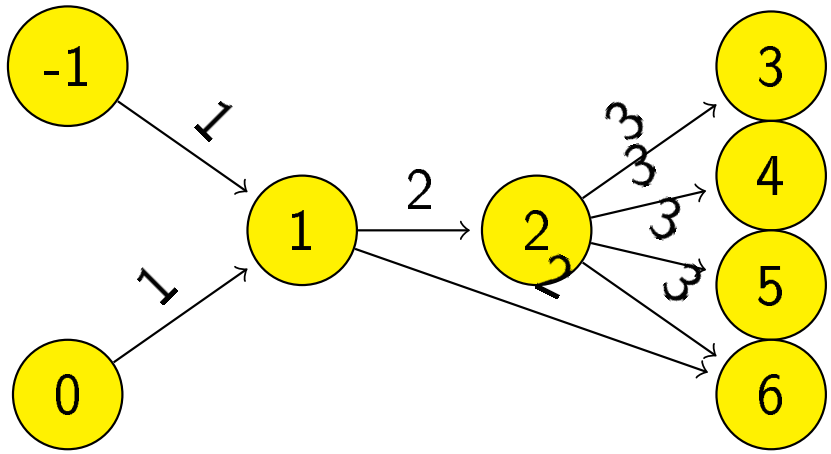
$(i,j) \prec^+ (\tilde{i},\tilde{j}) \Rightarrow h_{i,j} \neq h_{\tilde{i},\tilde{j}}$

Pre-bracketing der $A_{i,j}$

- $\tilde{A}_k := \prod_{h_{i,j}=k} A_{i,j}$
- $F' = \prod_k \tilde{A}_k$

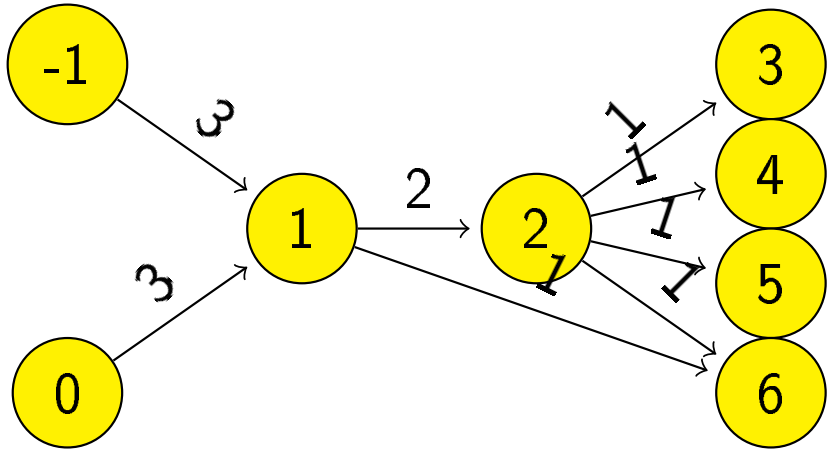
Nutze

- $\tilde{A}_k = \prod_{h_{i,j}=k} A_{i,j} = \sum_{h_{i,j}=k} A_{i,j} - (l_k - 1)Id_q$
- $l_k = \#\{(i,j) \in \mathcal{E} : h_{i,j} = k\}$

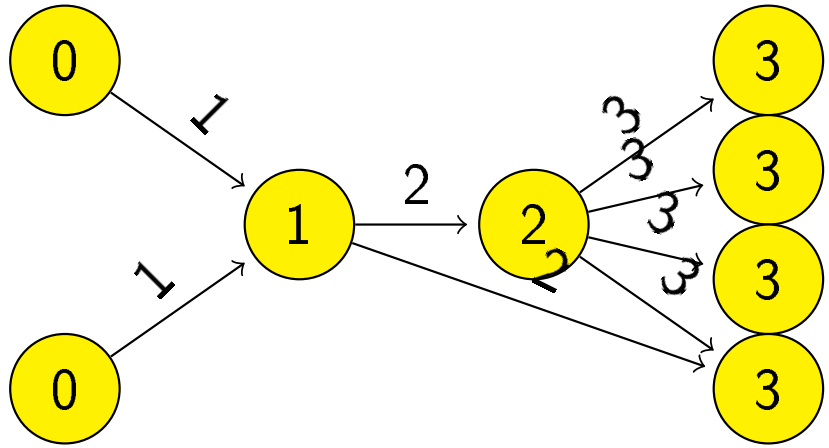


$$\tilde{A}_1 = A_{1,0} + A_{1-1} - Id_q$$
$$\tilde{A}_2 = A_{2,1} + A_{6,1} - Id_q$$
$$\tilde{A}_3 = A_{3,2} + A_{4,2} + A_{5,2} + A_{6,2} - 3Id_q$$

- Tiefe anstelle von Hoehe:



- Ecken anstelle von Kanten und A_i anstelle der $A_{i,j}$:



Numerische Ergebnisse

	n	l	m	DF	NR	DP	NR/DP
FDC	16	984	16	16000	11000	930	11.83
FCH	32	1209	32	39712	11169	845	13.22
WAT	7	1683	7	11830	11830	4240	2.79
DIE	20	2499	20	50380	50380	1659	30.34
VDI	100	504	100	60400	60400	10301	5.86
GDF	11	1625	65	18590	18590	1430	13.0

- DF = "Dense Forward Mode"
- NR = Methode von Newsam und Ramsdell
- DP = "Dynamic Programming"-ansatz

Zusammenfassung

- Berechung der Jakobi Matrix $\triangleq$ Produkt vieler dünnbesetzter Matrizen
- Steigerung der Effizienz durch
  - Umordnen der Matrizen
  - Pruning = Löschen nicht benötigter Zeilen und Spalten
  - Optimales Klammern
- Pre-bracketing um Aufwand bei der Suche nach einer optimal Klammerung ($= O(p^3)$) zu verringern

$\longrightarrow$ drei- bis sechsmal weniger Aufwand zur Berechnung von F' als bei den bislang besten Verfahren

Literatur


Andreas Griewank and Uwe Naumann.
Accumulating Jacobians as chained sparse matrix products.
Math. Program., 95(3, Ser. A):555–571, 2003.